University of Cyprus

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

# STEREO VISION HARDWARE ARCHITECTURES FOR REAL-TIME DEPTH COMPUTATION IN EMBEDDED VISION APPLICATIONS

DOCTOR OF PHILOSOPHY DISSERTATION

CHRISTOS TTOFI

2014

University of Cyprus

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING

# STEREO VISION HARDWARE ARCHITECTURES FOR REAL-TIME DEPTH COMPUTATION IN EMBEDDED VISION APPLICATIONS

CHRISTOS TTOFI

A Dissertation Submitted to the University of Cyprus in Partial Fulfillment
of the Requirements for the Degree of Doctor of Philosophy

May 2014

# VALIDATION PAGE

**Doctoral Candidate: Christos Ttofi**

**Doctoral Thesis Title: Stereo Vision Hardware Architectures for Real-Time Depth Computation in Embedded Vision Applications**

*The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the* ***Department of Electrical and Computer Engineering*** *and was approved on May 23, 2014 by the members of the* ***Examination Committee****.*

**Examination Committee:**

**Committee Chair** ΠΡΟΣΩΠΙΚΑ ΔΕΔΟΜΕΝΑ
Dr. Maria K. Michael

**Research Supervisor** ΠΡΟΣΩΠΙΚΑ ΔΕΔΟΜΕΝΑ
Dr. Theocharis Theocharides

**Committee Member** ΠΡΟΣΩΠΙΚΑ ΔΕΔΟΜΕΝΑ
Dr. Dimitrios Soudris

**Committee Member** ΠΡΟΣΩΠΙΚΑ ΔΕΔΟΜΕΝΑ
Dr. Constantinos S. Pattichis

**Committee Member** ΠΡΟΣΩΠΙΚΑ ΔΕΔΟΜΕΝΑ
Dr. Chrysostomos Nicopoulos

# DECLARATION OF DOCTORAL CANDIDATE

The present doctoral dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

………***Christos Ttofi***……………….. [Full Name of Doctoral Candidate]

….. **ΠΡΟΣΩΠΙΚΑ ΔΕΔΟΜΕΝΑ** ………….. [Signature]

# ΠΕΡΙΛΗΨΗ

Η ανάκτηση του βάθους, δηλαδή της απόστασης ενός αντικειμένου από μία διάταξη καμερών, μέσω υπολογιστικών συστημάτων στερεοσκοπικής όρασης, αναμένεται να οδηγήσει σε μια ραγδαία ανάπτυξη νέων εφαρμογών στα ενσωματωμένα συστήματα όρασης μηχανής, των οποίων η λειτουργία μέχρι τώρα βασιζόταν σε δισδιάστατη πληροφορία. Απτές εφαρμογές αυτής της νέας τάσης ήδη υπάρχουν σε πολλές περιοχές: από ηλεκτρονικά είδη ευρείας κατανάλωσης και ψυχαγωγίας, μέχρι συστήματα ρομποτικής, αυτοκινήτων, ιατρικής απεικόνισης, στρατιωτικές συσκευές κ.λ.π. Η στερεοσκοπική όραση μηχανής αποτελεί μια κατάλληλη τεχνολογία για την εκτίμηση πληροφορίας βάθους σε μια οπτική σκηνή, χρησιμοποιώντας στερεοσκοπικά ζεύγη εικόνων. Η διαδικασία αυτή επιτυγχάνεται με την αναζήτηση και εντοπισμό αντίστοιχων προβολών κοινών σημείων στο χώρο, τα οποία όμως ανιχνεύονται από τις δύο κάμερες σε διαφορετικές θέσεις. Σε γενικές γραμμές, η διαδικασία της στερεοσκοπικής αντιστοίχισης συνιστά ένα υπολογιστικά απαιτητικό στόχο και έχει επιλυθεί με διάφορα είδη αλγορίθμων, παράγοντας διαφορετικά αποτελέσματα όσο αφορά την ακρίβεια των αποτελεσμάτων και την υπολογιστική πολυπλοκότητα. Επιπρόσθετα, οι απαιτήσεις είναι ακόμα μεγαλύτερες σε εφαρμογές ενσωματωμένων συστημάτων, όπου το κόστος σε κατανάλωση υλικού, διαθέσιμης μνήμης και ενέργειας χρειάζεται να ελαχιστοποιηθούν.

Η παρούσα διατριβή διερευνά την αρχιτεκτονικές υλικού αλγορίθμων στερεοσκοπικής όρασης που έχουν τη δυνατότητα να ικανοποιήσουν τις απαιτήσεις ενσωματωμένων εφαρμογών όρασης μηχανής. Αρχικά, η διατριβή παρουσιάζει τη σχεδίαση μιας αρχιτεκτονικής στερεοσκοπικής όρασης, η οποία μέσω της ενσωμάτωσης ανίχνευσης ακμών σε εικόνες επιδιώκει να επιταχύνει τη χρονοβόρα διαδικασία αντιστοίχισης κοινών σημείων στις δύο εικόνες, με τη μείωση του συνολικού χώρου αναζήτησης, που έχει ως αποτέλεσμα τη σημαντική βελτίωση της επεξεργαστικής ταχύτητας. Η ενσωμάτωση ανιχνευτών ακμών μειώνει επίσης τους απαιτούμενους πόρους σε υλικό και μνήμη, δίνοντας έτσι τη δυνατότητα σχεδιασμού μιας παράλληλης, επεκτάσιμης και αποδοτικής ως προς τους απαιτούμενους πόρους αρχιτεκτονική που είναι σε θέση να επεξεργάζεται στερεοσκοπικές εικόνες υψηλής ανάλυσης σε πραγματικό χρόνο.

Στη συνέχεια, η διατριβή εστιάζεται στο σχεδιασμό της αρχιτεκτονικής ενός πολύπλοκου, αλλά με μεγάλη ακρίβεια, αλγορίθμου στερεοσκοπικής αντιστοίχισης που χρησιμοποιεί προσαρμοστικά βάρη υποστήριξης και κατάτμηση εικόνας, σε μια προσπάθεια να βελτιώσει την αξιοπιστία της διαδικασία αντιστοίχισης και να ικανοποιήσει τις εξαιρετικά υψηλές απαιτήσεις ακρίβειας αναδυόμενων ενσωματωμένων εφαρμογών όρασης μηχανής. Η διατριβή εισάγει βελτιστοποιήσεις που στοχεύουν στην προσαρμογή του αλγορίθμου, ώστε αυτός να μπορεί να υλοποιηθεί αποδοτικά σε υλικό. Η αρχιτεκτονική που προκύπτει επιτυγχάνει ένα αποτελεσματικό συμβιβασμό ταχύτητας/ακρίβειας σε σύγκριση με ήδη υπάρχοντα συστήματα αντιστοίχισης, ωστόσο, καταναλώνοντας μεγάλο ποσοστό διαθέσιμων πόρων.

Κατά συνέπεια, η διατριβή επικεντρώνεται στη συνέχεια σε μια εναλλακτική μέθοδο η οποία στηρίζεται στην υλοποίηση της στερεοσκοπικής αντιστοίχισης με τη χρήση του πρόσφατα προτεινόμενου καθοδηγούμενου φίλτρου εικόνας. Η διατριβή παρουσιάζει μια συμπαγής και αποδοτική σχεδίαση του φίλτρου σε υλικό, και αναδεικνύει τις δυνατότητες του φίλτρου για μείωση της πολυπλοκότητας της διαδικασίας στερεοσκοπικής αντιστοίχισης που είναι βασισμένη σε προσαρμοστικά βάρη υποστήριξης, αλλά και την αποτελεσματικότητά του για υλοποίηση μιας ισχυρής μονάδας βελτίωσης των εξαγόμενων πινάκων βάθους, η οποία μπορεί να βελτιώσει την ακρίβεια του αλγορίθμου στερεοσκοπικής αντιστοίχισης σημαντικά, ακόμη και αν είναι ενσωματωμένη σε απλούς αλγορίθμους αντιστοίχισης.

Τέλος, η διατριβή παρέχει ενδιαφέρουσες πληροφορίες που προκύπτουν από την αξιολόγηση των προτεινόμενων αρχιτεκτονικών στερεοσκοπικής όρασης, σε εφαρμογές ανίχνευσης αντικειμένων σε εικόνες, καθώς και αποφυγής εμποδίων σε αυτόνομα ρομποτικά συστήματα.

**Λέξεις Κλειδιά:** Ενσωματωμένα Συστήματα, Υπολογιστική Όραση, Ενσωματωμένη Όραση Μηχανής, Στερεοσκοπική Όραση, Πίνακες Αναδιατασσόμενης Λογικής

# ABSTRACT

Empowering embedded vision systems with 3D perception capabilities is expected to lead to a boost of new applications that so far could not be done with classical 2D alternatives. Applications of this new trend already exist in numerous areas: from consumer electronics and entertainment, to robotics, automotives, medical imaging, defense, etc. Stereo vision is a well-suited technology that uses two standard image cameras to infer depth information, by solving the so-called stereo matching problem. This involves searching and locating corresponding projections of the same 3D points sensed by the two cameras in different positions, a challenging task that can be tackled with many algorithms, consequently producing different outcomes in terms of accuracy and computational complexity. Stereo matching becomes even more challenging when targeting applications in embedded and mobile environments, where cost, energy and memory overheads need to be minimized.

This thesis investigates hardware architectures of stereo matching algorithms that have the potential to satisfy the requirements of constrained embedded vision applications. Initially, the design of a stereo matching architecture that utilizes edge information as a means to accelerate the overall matching process, is presented. By constraining the matching process only on binary data (edges), the search space is greatly reduced and the overall frame-rate is improved. The integration of edge information also reduces the logic and memory requirements, thus enabling the design of a parallel, scalable and resource-optimized architecture that is able to process HD stereo images in real time. Afterwards, the thesis focuses on designing the architecture of a complex, accurate matching algorithm that uses adaptive support weights (ADSW) and image segmentation, in an attempt to improve the robustness of the matching process and satisfy the extremely high matching accuracy required by many of today’s embedded vision applications. The thesis introduces hardware design optimizations to adapt the segment-based ADSW algorithm for a hardware-friendly and compatible with embedded constraints design. The resulting architecture obtains an effective speed-accuracy tradeoff when compared to state-of-the-art stereo matching systems, however at the expense of high resource usage. Consequently, an alternative method that implements stereo matching based on the recently proposed guided filter, is investigated. The thesis presents a compact and efficient design of the filter, and illustrates its potential in reducing the complexity of the ADSW matching process, but also its efficiency in enabling a powerful disparity refinement unit, which can improve the matching accuracy considerably, even if it is integrated into simple stereo matching algorithms. Finally, the thesis provides insights obtained from evaluating the proposed architectures in object detection and obstacle avoidance applications.

**Keywords:** Embedded Systems; Computer Vision; Embedded Vision; Stereo Vision; Real-Time; Field Programmable Gate Arrays (FPGAs)

# PUBLICATIONS

## Publications Stemming from this Thesis

### Journal Publications:

1. **Christos Ttofis**, Christos Kyrkou and Theocharis Theocharides, "A Hardware-Efficient Architecture for Accurate Real-Time Disparity Map Estimation," *ACM Transactions on Embedded Computing Systems,* accepted for publication, 2014.

2. Christos Kyrkou, **Christos Ttofis**, and Theocharis Theocharides, “A Hardware Architecture for Real-Time Object Detection Using Depth and Edge Information,” *ACM Transactions on Embedded Computing Systems***,** vol. 13, no. 3, pp. 54:1-54:19, December 2013.

3. **Christos Ttofis**, Stavros Hadjitheofanous, Athos Georghiades and Theocharis Theocharides, “Edge-Directed Hardware Architecture for Real-Time Disparity Map Computation,” *IEEE Transactions on Computers,* vol. 62, no. 4, pp. 690-704, April 2013.

4. **Christos Ttofis** and Theocharis Theocharides, “Hardware Design Considerations for Edge-Accelerated Stereo Correspondence Algorithms,” *VLSI Design – Special Issue on Circuits and Systems for Advanced Video Compression Standards,* vol. 2012, Article ID 602737, 17 pages, 2012.

### Publications in Refereed Conference Proceedings:

5. **C. Ttofis** and T. Theocharides, “High-Quality Real-Time Hardware Stereo Matching based on Guided Image Filtering,” *Design, Automation & Test in Europe Conference & Exhibition (DATE’14)*, Dresden, Germany, 24-28 March 2014.

6. **C. Ttofis**, D. Stavrou, D. Koukounis, T. Theocharides and C. Panayiotou, “A Laboratory Course on 3D Vision for Robotic Applications,” *Proceedings of the IEEE International Conference on Microelectronic Systems Education 2013 (MSE '13),* pp. 21-24, 2-3 June 2013, Austin, Texas, USA.

7. **C. Ttofis** and T. Theocharides, “Towards Accurate Hardware Stereo Correspondence: A Real-Time FPGA Implementation of a Segmentation-Based Adaptive Support Weight Algorithm,” *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE’12)*, Dresden, Germany, 12-16 March 2012.

8. C. Kyrkou, **C. Ttofis**, T. Theocharides, “FPGA-Accelerated Object Detection using edge information”, *Proceedings of Field Programmable Logic and Application Conference (FPL) 2011*.

9. C. Kyrkou, **C. Ttofis**, T. Theocharides, “Depth-directed hardware object detection,” *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE) 2011*, pp.1-6, Grenoble, France, 14-18 March 2011.

10. S. Hadjitheophanous, **C. Ttofis**, A. S. Georghiades, and T. Theocharides, “Towards Hardware Stereoscopic 3D Reconstruction: A Real-Time FPGA Computation of the Disparity Map,” *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE) 2010*, pp.1743-1748, Dresden, Germany, 8-12 March 2010.

Other Publications:

11. **Christos Ttofis**, "Disparity Estimation Hardware Architectures and Design Techniques for Embedded Stereo Vision Applications," *presented as part of PhD dissertation in PhD Forum at the IEEE Design Automation and Test in Europe (DATE’13) Conference*, Grenoble, France, March 2013.

Under Submission:

12. Martinianos Papadopoulos**,** **Christos Ttofis**, Christos Kyrkou, Theocharis Theocharides, “Real-Time Obstacle Avoidance for Mobile Robots via Stereoscopic Vision Using Reconfigurable Hardware,” *24th International Conference on Field Programmable Logic and Applications*, 2014
13. **Christos Ttofis** and Theocharis Theocharides, “A low-cost embedded accurate stereo vision system based on guided image filtering”, *Computer Vision and Image Understanding,* Elsevier, 2014.

## Other Publications by the Author

Journal Publications:

14. Dimitris Koukounis, **Christos Ttofis**, Agathoklis Papadopoullos, Theocharis Theocharides, “A High Performance Hardware Architecture for Portable, Low-Power Retinal Vessel Segmentation,” *Integration, The VLSI Journal, Elsevier,* to appear, 2014.
15. **Christos Ttofis**, Theocharis Theocharides, and Maria K. Michael, “FPGA-based Laboratory Assignments for NoC-based Manycore Systems,” *IEEE Transactions on Education*, vol. 55, no. 2, pp. 180-189, May 2012.
16. **Christos Ttofis**, Agathoklis Papadopoulos, Theocharis Theocharides, Maria K. Michael, and Demosthenes Doumenis, “An MPSoC-Based QAM Modulation Architecture with Run-Time Load-Balancing,” *EURASIP Journal on Embedded Systems,* vol. 2011, Article ID 790265, 15 pages, 2011.

Publications in Refereed Conference Proceedings:

17. D. Koukounis, **C. Ttofis**, T. Theocharides, “Hardware Acceleration of Retinal Blood Vasculature Segmentation”, *Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI (GLSVLSI'13)*, pp. 113-118, Paris, France, 2-3 May 2013.

18. **C. Ttofis**, A. Papadopoulos, T. Theocharides, M. Michael, D. Doumenis, "A reconfigurable MPSoC-based QAM modulation architecture," *Proceedings of 18th IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC 2010),* pp.137-142, 27-29 Sept. 2010.

19. **C Ttofis**, C. Kyrkou, T. Theocharides and M. K. Michael, "FPGA-Based NoC-Driven Sequence of Lab Assignments for Manycore Systems," *Proceedings of the IEEE International Conference on Microelectronic Systems Education 2009 (MSE '09),* pp.5-8, San Francisco, USA, 25-27 July 2009 - **Best Paper Award**

20. **C. Ttofis**, **T. Theocharides**, "A C++ Simulator for Evaluting NoC Communication Backbones," *in the Proceedings of the 3rd Greek National Student Conference of Electrical and Computer Engineering*, page 54, Thessaloniki, Greece, April 2009.

# ACKNOWLEDGEMENTS

During the three years of this thesis research work, I received much support from many people around me. Therefore, I would like to take this opportunity to express my gratitude and appreciation to these people. This thesis would not have been possible without their invaluable help and support.

First of all, I would like to thank my very passionate advisor, Dr. Theocharis Theocharides, for his guidance and support over these years. He have believed in me from the beginning, guided me steadily through this period, and managed to keep me calm in demanding and stressful situations. Furthermore, I will always be thankful for his knowledge, insistence, and the fact that he has provided a productive and friendly environment for research, factors that all strengthened this work significantly. It has been an honor to work with him.

I would also like to express my sincere thanks to my colleagues from the EASOC laboratory for the pleasant collaborations, as well as the enjoyable group meetings and discussions we had. A special thanks to my colleague Christos Kyrkou for his committed collaboration in the development of the "depth-accelerated hardware object detection system". Our collaboration has resulted in several publications, which are included in this thesis. I also thank my colleagues at the KIOS Research Center for establishing a nice and enjoyable working atmosphere.

During the three-year period of my PhD research, I was lucky to have the chance to work in the RUNNER project, together with a group of brilliant experts, who contributed in making meetings, visits and workshops a fun and rewarding experience. I am also indebted to the members of my examination committee professor Constantinos Pattichis, Dr. Maria Michael, Dr. Dimitrios Soudris, Dr. Theocharis Theocharides and Dr. Chrysostomos Nicopoulos, who dedicated time to review this dissertation, and provided feedback and suggestions.

On a personal note, I am so thankful to my wonderful family for the continuous support, patience and encouragement during the progress of my thesis. I am grateful for my wife, Ioanna, who always make me keep trying to reach my dreams, and for my son, Savvas, because he has brought me so much joy, but in addition to the smiles and laughter he brings, he has taught me more than I believe could have been possible. I now understand why they both came in my life so early. I am also thankful for my pets, which make life so much pleasanter. Lastly, I am thankful for God, who not only gave me all the above, but who is always in my corner, seeing me through each day!!

Thank You All

Christos Ttofi
May 23, 2014

## DEDICATION PAGE

***To My Family***

My Wife Ioanna Filippou, and

My son Savvas Ttofi

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

**(In Alphabetical Order)**

| | |
|---|---|
| 2D | Two Dimensional |
| 3D | Three Dimensional |
| ABDIF | Absolute Difference |
| AD | Absolute Difference |
| ADSW | Adaptive Support Weight |
| ALL | All Image Regions |
| API | Application Programming Interface |
| ASIC | Application Specific Integrated Circuit |
| BP | Belief Propagation |
| BTA | Binary Tree Adders |
| CBE | Cell Broadband Engine |
| CCU | Cost Computation Unit |
| CE | Classification Engine |
| CONV | Convolution |
| CPU | Central Processing Unit |
| CS | Calculation Stage |
| CVC | Cost Volume Construction |
| CVCU | Cost Volume Construction Unit |
| CVF | Cost Volume Filtering |
| CVFDSU | Cost Volume Filtering Disparity Selection Unit |
| CW | Constant Window |
| DCU | Disparity Computation Unit |
| DCU | Disparity Calculation Unit |
| DEU | Disparity Extraction Unit |
| DISC | Regions with Depth Discontinuities |
| DMA | Direct Memory Access |
| DP | Dynamic Programming |
| DRU | Disparity Refinement Unit |
| DS | Disparity Selection |
| DSI | Disparity Space Image |
| DSP | Digital Signal Processor |
| EDK | Embedded Development Kit |
| EDU | Edge Detection Unit |
| ETU | Edge Tracking Unit |
| FIFO | First-In First-Out |
| FPGA | Field Programmable Gate Array |
| FPS | Frames per Second |
| FSL | Fast Simplex Link |
| FW | Fixed Window |
| GCC | Gradient Computation Core |
| GCMMU | Gradient Computation Memory Management Unit |
| GIF | Guided Image Filter |

| | |
|---|---|
| GPP | General Purpose Processor |
| GPU | Graphics Processing Unit |
| HD | High Definition |
| HDL | Hardware Description Language |
| I/O | Input / Output |
| IMU | Input Management Unit |
| IP | Intellectual Property |
| IR | Infrared |
| IS | Input Stage |
| LoG | Laplacian of Gaussian |
| LR-check | Left-Right Consistency Check |
| LS | Local Store |
| LUT | Look-up Table |
| MA | Memory Arrangement |
| MCCU | Memory Controller Control Unit |
| MDE/s | Million Disparity Estimations per second |
| MFC | Memory Flow Controller |
| MRF | Markov Random Field |
| MUL_ADD | Multiplication-Addition |
| MW | Multiple Window |
| NCC | Normalized Cross Correlation |
| NN | Nearest Neighbor |
| NON_OCC | Non Occluded Regions |
| NORM | Normalization |
| NRE | Non Recursive Engineering |
| PLB | Processor Local Bus |
| PPE | PowerPC Processor Element |
| PPU | Power Processing Unit |
| PWM | Pulse Width Modulation |
| RMS | Root-Mean-Square |
| RoI | Region of Interest |
| RPi | Raspberry Pi |
| RSB | Reference Scanline Buffer |
| SAD | Sum of Absolute Difference |
| SC | System Configuration |
| SCV | Stereo Cost Volume |
| SDK | Software Development Kit |
| SGM | Semi Global Matching |
| SHD | Sum of Hamming Distances |
| SIMD | Single Instruction Multiple Data |
| SMT | Simultaneous Multithreading |
| SNR | Signal to Noise |
| SoC | System on a chip |
| SPE | Synergistic Processing Element |
| SSD | Sum of Square Differences |
| SSE | Streaming SIMD Extensions |

| | |
|---|---|
| SVM | Support Vector Machine |
| SXU | Synergistic eXecution Unit |
| TAD | Truncated Absolute Difference |
| TFT | Thin Film Transistor |
| ToF | Time of Fight |
| TSB | Target Scanline Buffer |
| UHD | Ultra High Definition |
| USB | Universal Serial Bus |
| VLIW | Very Long Instruction Word |
| VW | Varying Window |
| WEU | Window Extraction Unit |
| WMF | Weight Median Filtering |
| WTA | Winner-Takes-All |

# CHAPTER 1

# Introduction to Embedded Vision and Depth Perception - Thesis Motivation & Contributions

*THE development of a stereo vision system comprises an interdependent design process, where the efficiency of the resulting system depends both on the hardware architecture and the associated algorithms. The design of such a system becomes even more challenging when targeting applications in embedded and mobile environments, where cost, energy and memory overheads need to be minimized. This thesis investigates hardware architectures of stereo vision algorithms that have the potential to satisfy the requirements of constrained embedded vision applications. This chapter has two purposes. First, it provides an overview on the development and deployment of embedded vision / stereo vision technology into a variety of interesting and promising applications, highlighting advances in enabling technologies, including processors, sensors, and development platforms. Second, it motivates the thesis background, and summarizes its main contributions and the individual chapters.*

## 1.1 The Embedded Vision Revolution

Most of us have smart phones and tablets with front- and rear-viewing cameras capable of capturing high-resolution still images and high-definition video clips. Many of us have enjoyed the fresh new gaming experiences offered by the Xbox's Microsoft Kinect and Sony's Playstation 4 video game consoles. Some of us may even have a car with a rear-view parking camera, or a more advanced driver assistance system capable of detecting pedestrians, lanes, or even classify various traffic signs including speed limit signs, etc. What we may not realize is that all of these devices, which have recently become an essential part of our everyday lives, have something in common – *embedded vision*.

Embedded vision is a technology that entails a hybrid of two well-established fields: Embedded Systems and Computer Vision. This emerging technology aims at incorporating automated image analysis and vision capabilities into any kind of a computer-based system that is not a general purpose computer, but rather it is designed to perform certain tasks only. Using digital processing and intelligent algorithms, an embedded vision system can extract and interpret meaning from images or video, enabling it to understand the surrounding world and interact with its host environment [**1**]. Embedded vision can lead to the development of safer, smarter and more responsive machines, which like humans, see and understand. To put it simply, embedded vision refers to devices or machines that are empowered with the gift of sight, and are able to see and understand their environment!

Although computer vision algorithms have been extensively studied over the last few decades in academic research, they have only been implemented using large, heavy, expensive, and power-draining computers, restricting their usage to a short list of applications such as factory automation / assembly line inspection, optical character recognition and military systems [**2**]. In recent years, however, the emergence of very powerful, low-cost and energy-efficient processors, image sensors, memories, and other semiconductor devices, along with robust computer vision algorithms, has made embedded vision much more accessible and feasible [**3**]. Nowadays, even inexpensive smart phones and tablets are capable to supply formidable processing capabilities, including multi-core high-frequency CPUs and embedded graphics processors, on-chip DSPs and imaging coprocessors, and multiple gigabytes of memory. They are also provided with front- and rear-viewing camera sensors that support high image resolutions and frame rates. Therefore, a major transformation is underway, aiming to integrate vision capabilities into a wide variety of embedded systems and electronic products to make them more intelligent and responsive than before, and thus more valuable to the users.

An embedded vision system is comprised of four major elements [**4**], which are illustrated in Figure 1.1. The image sensor outputs images at some resolution of pixels and a specific rate, which corresponds to how many image frames the sensor outputs per second. These images are processed by an embedded processing device, in the form of specialized processors that implement unique architectures, or dedicated accelerators specific to image and video processing. Image sensors generate image/video data in a streaming fashion. Processing the data output from the video sensor usually requires storing in memory either all or some parts of the image/video. Finally, specialized vision algorithms are required to manipulate and analyze the vast amount of incoming video data to extract visual meaning about the surrounding 3D world.

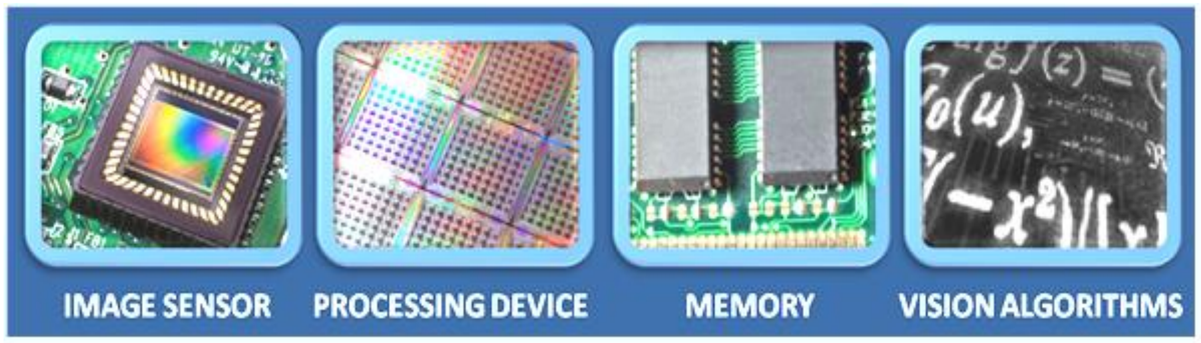


**Figure 1.1: Elements of an embedded vision system.**

Due to continues technological advances in sensors, processors, memory and algorithms, embedded vision systems have nowadays the potential to revolutionize a multitude of industries, including medicine, advertising, security, personal health, entertainment, automotive and more [**5**]. Embedded vision has high potential in medicine, where it can be incorporated for example in medical electronic devices such as intelligent x-ray and MRI systems to assist radiologist to rapidly and accurately identify image irregularities, eliminating degrading factors like fatigue and distraction, which occurred when the image analysis is performed by humans [**6**]. Another medical application involves the detection of skin cancer signs in moles on the human body, using a smartphone to capture images of a mole and process them by a complex vision algorithm developed by dermatologists [**6**]. Other revolutionary medical applications aim at providing assistance to blind people, by utilizing a camera to interpret real objects and communicate them to the user as auditory cues [**7**]. In automotive, vision-based systems utilize gesture and face recognition for car safety; the driver for example can use a winking of the eye to turn the radio on and off, or a movement of the head to change the volume, thus reducing distractions while driving [**7**]. Furthermore, the ability of such systems to detect meaning from images of the road ahead the car could be used to provide warnings if for example a car begins leaving a lane, approaches a car too closely, or detects a bicycle or a pedestrian. Furthermore, active research in the field of embedded vision aims to incorporate face recognition for advertising, in order to track the facial responses of internet users while they view online advertisements [**7**]. In general, the era of embedded vision has just started. We would need several pages to list the abundant applications that could benefit from the use of this emerging technology, as the technology’s potential is fundamentally limited only by our imagination. Moreover, there are great expectations that within the next ten years, embedded vision will broaden and accelerate its penetration into numerous new markets, creating exciting products for a range of applications [**2**].

The rapid proliferation of embedded vision technology is considered similar to the evolution of high-speed wireless connectivity, which initially began as an exotic and costly technology found only in complex and expensive systems, but when digital circuits got fast enough, inexpensive enough, and energy efficient enough, it became a mass-market technology. Today, advances in digital integrated circuits are critical enough to pave the way for the proliferation of embedded vision into high volume applications, in virtually every category of electronic products, from automobiles to consumer electronics to health care, etc.

Embedded vision applications usually rely on advanced computer vision algorithms that typically require high processing performance. In addition, they are deployed in embedded systems of all kinds that need to fit into tight memory, power consumption, size, communication bandwidth and cost envelopes. In contrast to other digital signal processing application domains, such as wireless communications, where the aforementioned constraints are satisfied by the use of specialized accelerators and coprocessors implemented on Application Specific Integrated Circuits (ASICs), the use of non-programmable accelerators is less attractive for embedded vision applications. This is mainly attributed in the fact that there are no common standards available to constrain the choice of vision algorithms; there are usually many approaches to choose from to solve a particular vision problem. As such, the majority of computer vision algorithms are implemented on programmable embedded computing platforms such as Digital Signal Processors (DSPs), Field Programmable Gate Arrays (FPGAs), and several kinds of multi-core Central Processing Units (CPUs) [**2**]. These platforms though are based on heterogeneous and specialized architectures, therefore, to meet the challenging combination o high performance, low power and low cost, embedded vision systems are required to exploit the inherent parallelism found in such architectures. Efficient development methods are also an important issue.

Besides the general parameters of performance, cost and integration associated with the computing platform, there are also other critical parameters that restrict the computing devices that can be selected for implementing computer vision applications. Factors such as the way in which data are transferred to the device (I/O operations), the data type and representation, and the program flow, are all critical towards delivering revolutionary vision processing in performance-demanding and power-sensitive embedded vision applications [**8**]. Hence, an embedded vision system must be able to deal with all of them.

## 1.2 Embedded Stereo Vision - Motivation

Among the most important tasks of an embedded vision system is the extraction of depth information. This can become possible through stereo vision technology, which delivers precise depth information of the observed scene in a manner similar to human binocular vision [**9**], [**10**], [**11**]. It uses two cameras that capture the same scene from two different viewpoints, and depth information is inferred by finding corresponding points in the two images, a process known as stereo matching (or disparity estimation). Stereo vision has enabled the realization of applications, which so far could not be done with classical 2D computer vision technologies [**11**]. Many applications of stereo vision are in inspection, medicine, automotives, etc. [**11**]. Moreover, detailed 3D information about the surrounding world is important to enable autonomous navigation, obstacle detection and classification in unknown environments [**12**]. Therefore, stereo vision is also used in several consumer electronics and multimedia products that demand vision-based autonomous behaviors (e.g. mobile robots, intelligent surveillance and autonomous vehicles). One successful example is the UK's Oxford robot car, an autonomous car recently unveiled by a team from Oxford University that uses stereo vision to locate and track vehicles [**13**]. The new Mercedes-Benz S-class 2014 also features a stereoscopic camera that sees objects ahead in 3-D and helps to detect cross-traffic and pedestrians [**14**]. With products like these demonstrating what it is possible, several other application areas of stereo vision that span virtually every embedded market will experience huge growth rates in the near future.

While stereo vision is a rather popular and important subject in the fields of machine vision and image analysis, it is a computationally expensive task. Therefore, its use in consumer electronics and multimedia products has posed new challenges that need to be tackled. Such systems usually impose several, often contradictory constraints, including real-time processing, high matching accuracy and low power consumption. This makes the successful realization of an embedded stereo vision system a key challenge, both in terms of the matching algorithm, as well as in terms of the implementation platform. On one hand, implementations of complex stereo matching algorithms produce very accurate results but rely on the high-end hardware resources of multi-core CPUs and/or GPU platforms to achieve real-time processing. Such platforms therefore are unsuitable for the realization of stand-alone stereo matching systems, and also consume excessive power, which is not desirable in embedded vision applications, especially in the case of battery-operated devices. DSP platforms, on the other side, feature lower power consumption than software implementations running on high-end processors or GPUs, thus they are more favorable in embedded stereo vision applications. However, the fixed logic and small number of processing cores prohibit this platform from offering the required parallelism for real-time performance. Recent research indicates that application-specific hardware acceleration using either FPGAs or custom circuits (ASICs) might be the most appropriate solution for embedded stereo vision applications, since it can provide high computational power with low power consumption. Furthermore, it allows the architectures to be designed in a customized way; therefore, the computational resources can be optimized in terms of resource utilization. To this end, a lot of work has been carried out on real-time dedicated hardware implementations of disparity estimation algorithms on both FPGAs and ASICs. However, FPGAs remain the most popular implementation choice because of their inherent parallelism, re-programmability and great flexibility in manipulating the algorithm, and relatively short design cycle.

Application-specific hardware acceleration has been proven to be a promising means towards designing efficient stereo matching hardware architectures. Existing efforts towards this direction have yielded encouraging results, yet satisfying the hard real-time constraints of emerging embedded stereo vision applications, especially for High Definition (HD) images and under limited resource usage, is still challenging. Moreover, existing dedicated hardware implementations are mostly directed towards implementing simple stereo algorithms, therefore trading accuracy for speed. Hence, this research is concerned with the implementation of hardware architectures of different stereo matching algorithms, which can be used to satisfy the different constraints involved in embedded environments.

## 1.3 Objectives and Scope of Research - Thesis Contributions

The scope of this thesis is on dedicated hardware architectures of stereo matching algorithms for efficient 3D perception in embedded applications. Although there have been several attempts recently to implement real-time stereo vision systems on dedicated hardware, most of them cannot satisfy hardware efficiency and memory footprint requirements that characterize vision applications running on platforms with limited resources. As such, the first problem addressed in the thesis is how to find novel ways to satisfy the requirements of hard real-time, resource constrained embedded vision applications. The thesis proposes the design of a hybrid, block- and feature-matching stereo algorithm; through an edge detector that generates the features (edges) used to reduce the search space, and a block matching Sum of

Absolute Difference (SAD) algorithm used for the stereo matching computation. The integration of edge information constrains the stereo matching process only on binary data (edges), therefore reducing the search space and improving the overall frame-rate, while also reducing the logic and memory requirements, thus enabling the design of a parallel, scalable and resource-optimized architecture that is able to process HD stereo images in real time (50Hz@1280x1024). Furthermore, edges represent reliable image features, and their use reduces the sensitivity to pixel intensity variations caused by camera gain or illumination changes. Therefore, the proposed edge-directed architecture outperforms traditional SAD block matching-based hardware architectures in terms of matching quality. These features make the developed architecture suitable for resource-constrained embedded vision systems that need to satisfy hard real-time and low-power constraints.

However, many of today's applications not only need to satisfy embedded constraints but also the extremely high matching accuracy offered by more complex stereo algorithms. Hence, the second problem address in the thesis is the implementation of complex stereo matching algorithms through hardware-oriented algorithmic modifications, in an attempt to balance accuracy and speed in embedded stereo vision applications. To this end, the second part of the thesis presents the design of an Adaptive Support Weight (ADSW) algorithm that integrates information obtained from image segmentation in an attempt to improve the robustness of the matching process. The thesis also presents hardware-oriented algorithmic modifications and optimization techniques that make the algorithm hardware-friendly and compatible with embedded constraints. A prototype of the architecture was implemented on a Kintex-7 FPGA board, achieving 60 frames per second (fps) for 640x480 image sizes. At the same time, synthesis results targeting a commercial CMOS 65-nm cell library indicate that the architecture can be extended to larger scale systems.

The good quality results of the segment-based ADSW architecture come at the expense of high resource usage. As a result, a final but important objective of the thesis is the design of a stereo matching hardware system able to satisfy both speed and accuracy aspects with low resource requirements. To this end, the thesis focuses on the design of a fully pipelined, parallel and scalable stereo matching hardware architecture based on the recently proposed Guided Image Filter. This type of filter has been employed to reduce the complexity of the cost aggregation step in ADSW methods implemented in software. Therefore, the thesis aims to present a new and efficient hardware design of the GIF (that can be potentially adopted in

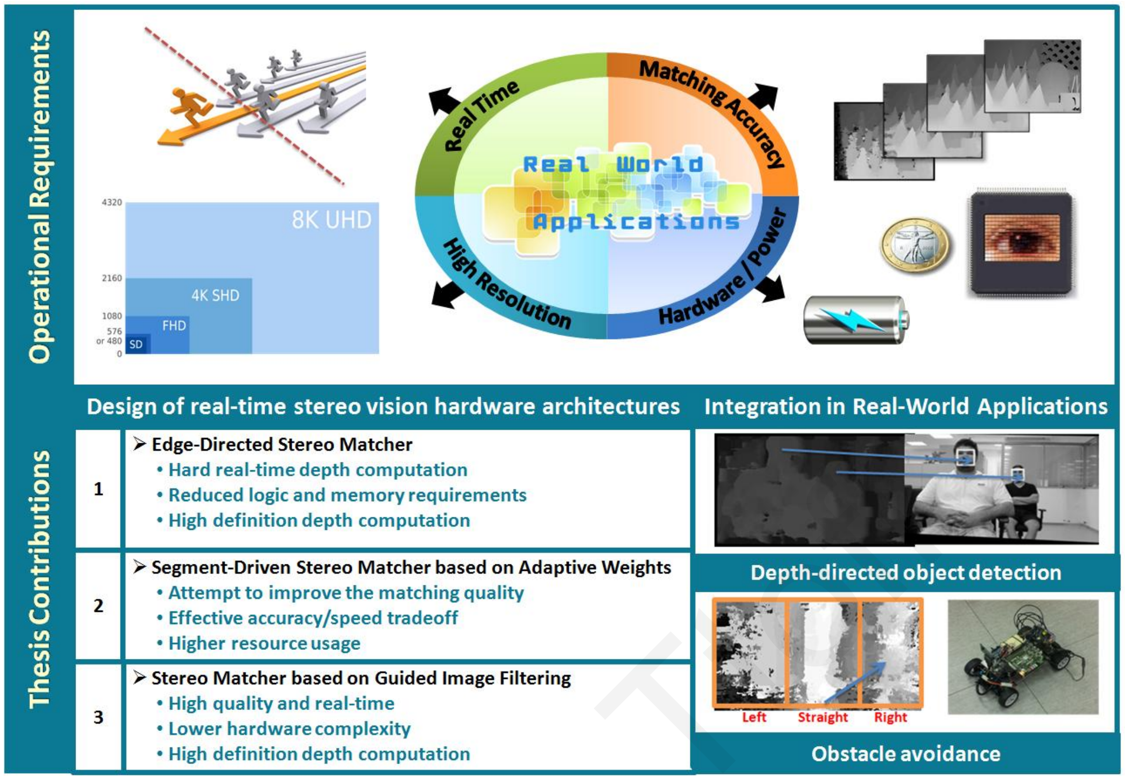

Operational Requirements
4320
2160
1080
576 or 480
0
8K UHD
4K SHD
FHD
SD
Real Time
Matching Accuracy
High Resolution
Hardware / Power
Real World Applications
Thesis Contributions
Design of real-time stereo vision hardware architectures
Integration in Real-World Applications
1
➢ Edge-Directed Stereo Matcher
• Hard real-time depth computation
• Reduced logic and memory requirements
• High definition depth computation
2
➢ Segment-Driven Stereo Matcher based on Adaptive Weights
• Attempt to improve the matching quality
• Effective accuracy/speed tradeoff
• Higher resource usage
3
➢ Stereo Matcher based on Guided Image Filtering
• High quality and real-time
• Lower hardware complexity
• High definition depth computation
Depth-directed object detection
Left
Straight
Right
Obstacle avoidance

**Figure 1.2: Scope of Research and Thesis Contributions.**

other uses of the filter), and also to explore and concurrently discuss the hardware design parameters and optimizations involved in integrating the GIF hardware architecture in the cost aggregation step of ADSW-based hardware stereo matching systems. The latter reduces the overall hardware complexity of cost aggregation, which in turn allows real-time stereo matching of HD images, as well as improvements of the overall matching accuracy, thanks to the edge-preserving property of the GIF.

Overall, the work in this thesis aims at providing depth discernment in embedded vision applications. Hence, the research done is related to the implementation of hardware architectures of stereo vision algorithms that have the potential to satisfy the requirements of constrained embedded vision applications. The developed architectures satisfy different performance parameters (operation requirements) in the content of embedded vision systems (optimized for the specific requirements of embedded environments), and can be utilized for the development of real-world vision applications. Figure 1.2 graphically illustrates the research purpose and scope of this thesis, and its technical contributions. The thesis contributions are also briefly summarized below:

- Initially, a stereo vision architecture that targets resource-constrained embedded applications with hard real-time and low-power constraints is designed. This architecture integrates an edge detection mechanism as a means to accelerate the overall matching process, and reduce its logic and memory requirements.

- The problem of improving the robustness of the hardware stereo matching process is then examined. Hence, a more complex, but accurate, stereo algorithm based on image segmentation and adaptive weights is also implemented. Through a series of hardware-oriented optimization techniques, the architecture enables an effective speed-accuracy tradeoff, however at the expense of high resource usage.
- Consequently, an alternative architecture that implements the stereo matching process based on the edge-preserving guided image filter is investigated. A compact and efficient design of the filter is utilized to reduce the hardware complexity of the ADSW matching process, while maintaining high-quality depth estimation results.
- In order to illustrate how the results achieved in the thesis can be used long term by the embedded vision community, and how the proposed stereo vision architectures can contribute significantly in the development of future 3D vision algorithms and applications, the thesis also provides insights obtained from evaluating the developed architectures in the applications of object detection and obstacle avoidance in robotics environments.

## 1.4 Overview of Chapters

The thesis is organized as follows. Chapter 2 provides fundamental concepts and background material related to the general problem of depth estimation, focusing primarily on the process of depth estimation through stereo vision technology and the associated matching algorithms. Chapter 2 also provides an overview of existing stereo vision implementations. Chapter 3 presents the research work that relates to the edge-directed stereo matching hardware architecture, and Chapter 4 presents the segmentation-driven stereo matching architecture that uses adaptive cost aggregation for improved accuracy. Chapter 5 investigates the possibility of utilizing the recently proposed guided image filter to design hardware-based stereo matching architectures able to provide high accuracy and concurrently high performance for embedded vision devices, with limited hardware and power budget. Evaluation of the proposed stereo matching architectures in real-world application scenarios are also presented in Chapter 6. Finally, Chapter 7 concludes with the most important findings, and provides future directives on how the work presented in the thesis can be improved.

## 1.5 The Main Contributions of the Author

The research that forms the basis of this dissertation has been conducted at the Embedded and Application Specific System-on-Chip (EASOC) Laboratory of the KIOS Research Center for Intelligent Systems and Networks, under the RUNNER - Reconfigurable Ultra Autonomous Novel Robots – project (co-funded by the Republic of Cyprus through the Research Promotion Foundation and the EUREKA Organization under the Eurostars Programme). In alignment with the objectives of the RUNNER project, the thesis aimed at utilizing high-end reconfigurable devices in order to allow for extremely high performance and power-efficient processing when implementing 3D sensing/matching, object recognition and obstacle avoidance schemes, with the objective to provide a framework that can be used to create highly autonomous robots with much better perception than the existing solutions. The author has contributed to the development of the robot's perception subsystem through the developed stereo vision architectures presented in Chapters 3-5. By taking advantage of the developed architectures and their ability to provide detailed information regarding the position and size of objects in the 3D space, a sophisticated object-detection scheme was also designed and implemented. The object detection system uses depth information to estimate the size of the objects in the environment of the robot, so as to identify with high accuracy and on real time (based on the reduction of the number of scales used in object detection) the objects around it. The author developed the object detection subsystem in collaboration with Christos Kyrkou, a colleague from the EASOC laboratory. Furthermore, the efficiency of the robot perception subsystem, together with its wide applicability was demonstrated in a real-world application scenario (obstacle avoidance in unconstrained indoor and outdoor environments).

# CHAPTER 2

# Stereo Vision for Depth Perception: Fundamentals and Relevant Work

*THE requirements of vision systems and products for emerging embedded applications, in the previous chapter, have clarified that the applications that benefit from 3D sensors are abundant. In this chapter, the objective is to cover fundamental concepts and background material related to the general problem of depth estimation, focusing primarily on the process of depth estimation through stereo vision technology. The chapter provides a description of the major steps involved in a stereo vision system, and serves as a partial literature overview of major stereo matching algorithms. The challenges involved in developing practical stereo vision systems for use in real-world applications are also discussed. This chapter finally looks into existing stereo vision implementations, and categorizes them with respect to their processing speed, matching accuracy and power consumption.*

## 2.1 A Brief Introduction to Computer Vision and Depth Estimation

A digital image is a projection of the three-dimensional (3D) scene from a specific viewpoint onto a two-dimensional (2D) plane [**15**]. During this process, the depth information is mathematically lost. The problem of *depth estimation* refers to the set of algorithms and techniques used to obtain a representation of the spatial (three-dimensional) structure and geometry of a scene [**9**]. This representation is required in many tasks in computer vision, robotic navigation, computer graphics, etc. In computer vision for example, extracting and utilizing depth information about the scene can bypass many traditional problems, such as varying foreground/background colors and textures, unknown object scales, etc. Considering the essential role that depth estimation plays is several applications, a wide variety of techniques have been proposed for acquiring the 3D geometry of objects in the scene, including stereo vision [**16**], laser scanning [**17**], time-of-flight [**18**], and structured-light [**19**]. This introductory section reviews basic concepts of depth estimation, covering background material related to image processing (e.g. image representation, different kind of images, etc.) and possible ways to represent 3D images over a plane, and also introducing testbeds and evaluation metrics used for the assessment and ranking of depth estimation methods.

### 2.1.1 Image Representation

The field of computer vision is concerned with extracting useful information about the three dimensional (3D) world from two dimensional (2D) images and video [**9**], [**20**]. Hence, the term computer vision is usually considered equivalent (or superset) of the term digital image analysis [**9**]. A computer vision system relies on visual input and image processing algorithms to emulate human vision, including learning and being able to make inferences and take actions, using computer software and hardware. This section is for those people who want to read this thesis and do not necessarily have any previous knowledge about image representation and analysis. Consequently, this section reviews basic concepts related to the representation of images in computer systems, like types of images, pixels, channels, resolution, etc.

Images are captured by cameras and stored digitally in computer memory or external disk. Typically, they are represented by means of a multidimensional matrix of binary numbers, and the value of each matrix element is called a *pixel*. A pixel (picture element) is the small block that represents the amount of color or gray intensity to be displayed for that particular portion of the image. The pixels of an image are arranged in forms of rows and columns. This is the most commonly found image type, called *raster image*. Each horizontal line in the image is called a *scanline*. The total number of pixels in an image is referred to as image *resolution*, and is usually represented in $HxW$ format, where $W$ is the width of the image and $H$ is the height of the image. *Aspect ratio* is basically a ratio of Width:Height of the image. Images can be represented in three ways – Color Images, Grayscale Image and Black and White Images (Binary Images), depending on the number of colors each pixel is comprised. All the aforementioned concepts are shown in a nutshell in Figure 2.1.

### 2.1.2 Color, Grayscale and Binary Images

Similar to the way humans perceive color through wavelength-sensitive sensory cells called cones, digital cameras use sensors that have different sensitivity to electromagnetic radiation (light) of different wavelength. One sensor is mainly sensitive to red light, one to green light, and one to blue light. It is the combination of the light sensitivities of the sensors (the three basic colors: red, green and blue) that determine the camera's actual color response.

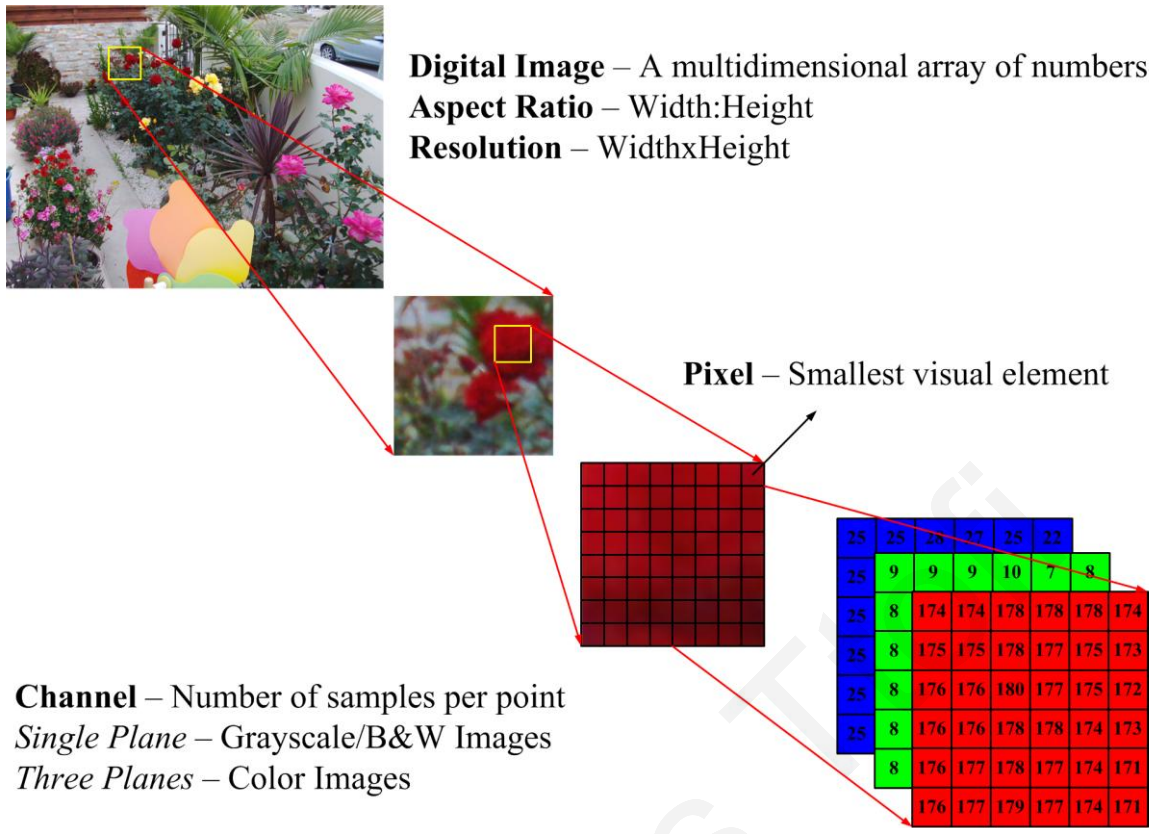

**Digital Image** – A multidimensional array of numbers
**Aspect Ratio** – Width:Height
**Resolution** – WidthxHeight
**Pixel** – Smallest visual element
**Channel** – Number of samples per point
*Single Plane* – Grayscale/B&W Images
*Three Planes* – Color Images
25 25 28 27 25 22
25 9 9 9 10 7 8
25 8 174 174 178 178 178 174
25 8 175 175 178 177 175 173
25 8 176 176 180 177 175 172
25 8 176 176 178 178 174 173
8 176 177 178 177 174 171
176 177 179 177 174 171

**Figure 2.1: Digital Image Representation in a Nutshell.**

Through these combinations, cameras can generate almost any perceivable color. A *color image* captured by a camera is usually processed by a computer or stored in memory, thus it is often represented as three separate image matrices; one storing the amount of red (R) in each pixel, one the amount of green (G) and one the amount of blue (B). This representation is called RGB format.

A *grayscale image* on the contrary does not differentiate the amount of sensitivity of the different colors. Rather, the same amount of color is perceived by each channel. As such, grayscale images are represented using a single image matrix, which differentiates the amount of light sensitivity for each pixel (pixel intensity), with dark pixels representing little light and bright pixels much light.

Another kind of image that utilizes only a single bit to represent each pixel is the *binary image*. Pixels in a binary image can only exist in two states, usually black or white, used to differentiate for example foreground from background pixels, image pixels located at object boundaries or not, etc. It must be noted that binary images are not provided directly by the camera sensor, but usually are the result of an image processing algorithm applied on a color or a grayscale image.

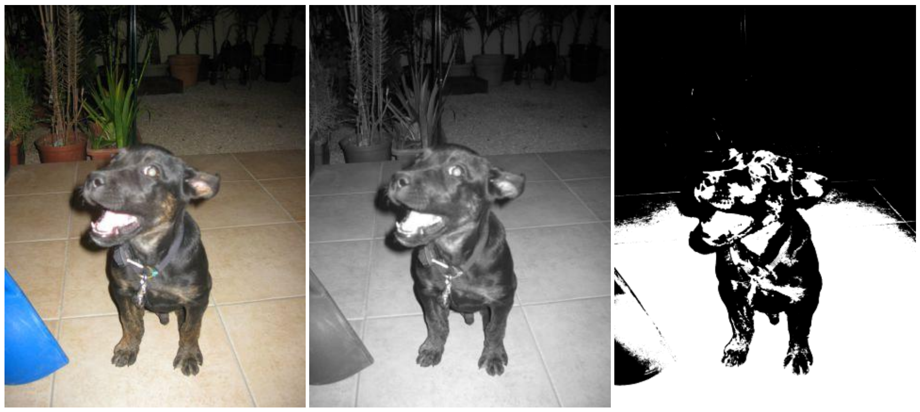


**Figure 2.2: Color vs. grayscale, vs. binary image.**

One of the first design choices in image processing and computer vision applications, whichever is the goal, is to decide what kind of images to process. Color images contain much more information (24 bits per pixel) compared to grayscale (8 bits per pixel), thus they should be more suitable for data extraction (e.g. depth). However, processing color images requires more time and calculations than processing grayscale images. Motivated by the fact that humans can understand images containing lines, silhouettes and other images formed using only two grey levels, binary images can also be used in many computer vision applications to reduce the amount of data to be processed, while providing an adequate representation of the scene. Their use is more essential when dealing with embedded and resource-constrained applications that require fast computations with less storage and energy consumption.

## 2.1.3 Representation of 3D images over a plane

### 2.1.3.1 *Grayscale 2.5D representation*

A color image stored in a computer system is the result of a quantization process that reduces the number of distinct colors used to represent the “analogue world”, so that it can be displayed and process on devices with limited number of colors / bits. As discussed in the previous subsection, an RGB image consists of three color channels (red, green and blue), which usually are represented by a matrix of unsigned integers of 8 bits (capable to represent 256 colors each one). In this way, the quantized image is visually similar as possible to the original image.

Similarly, the objects in the real-world can have an infinite number of possible depth values (analogue domain). As a result, the depth measured by a 3D camera sensor is discretized into parallel planes, with each plane representing a depth value as shown in Figure 2.3. Depth values are stored in a computer as a matrix of unsigned integers, the bit width of

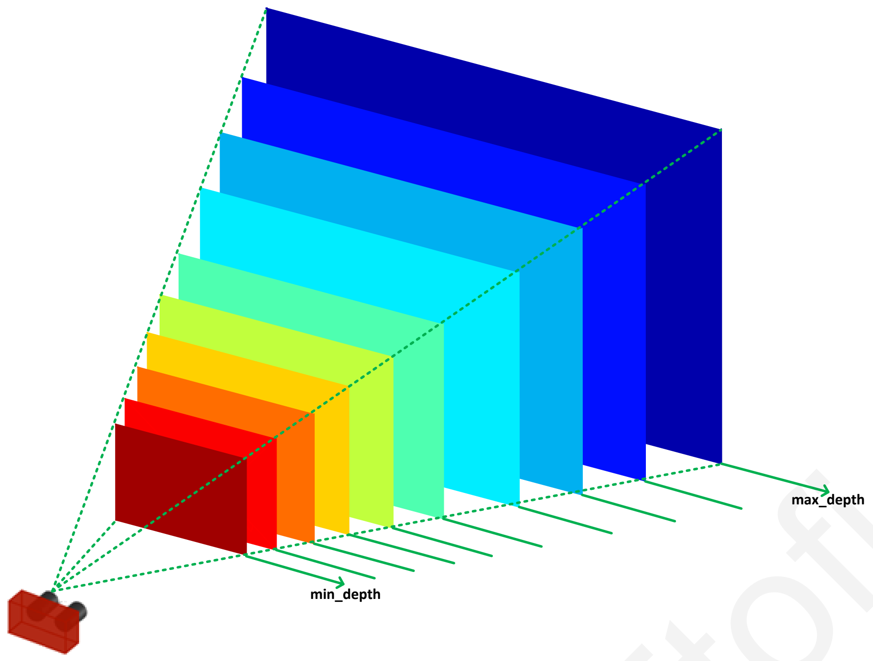


**Figure 2.3: Discetization of depth.**

which determines the precision of the obtained 3D data. The intensity of these integers represents the depth of each pixel in the original RGB image; however, the color, texture and luminosity of the original image are lost [**21**]. More specifically, the grayscale pixel intensities represent the inverse of the depth, with brighter pixels corresponding to points that are closer to the camera, and darker points corresponding to points that are further way from the camera. Figure 2.4 (a)-(b) shows an RGB image and its corresponding depth grayscale values (depth map). This is the most commonly used way for depth representation today, and is referred to as 2.5D representation, as the depth information is directly in each pixel, while it is represented on a 2D space. The resulting image data representing the 3D information is usually referred to as *range image*, *depth image*, *depth map*, *surface profiles*, or *2.5-D image*. We will use the term depth map throughout this thesis.

### 2.1.3.2 *Color 2.5D representation*

This depth representation is an extension of the previous one to the case where the depth values are represented by a color RGB image, rather than a grayscale one. An example of such a representation is shown in Figure 2.4 (c), where red-black colors represent closer points, and blue-dark colors the further points. This representation is more probably inspired by studies showing the effect of object color on depth ordering [**22**]. However, other color representations are also available in the literature (e.g. [**23**]).

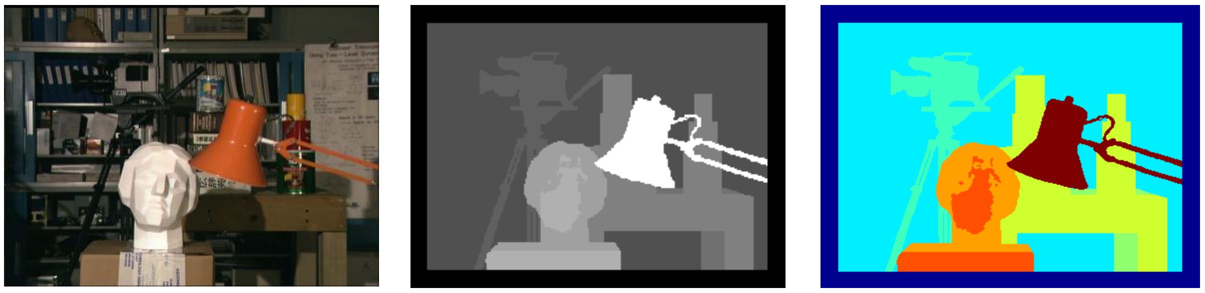

(a) (b) (c)

**Figure 2.4: Tsukuba (left view) and its grayscale and color-based 3D representation [25].**

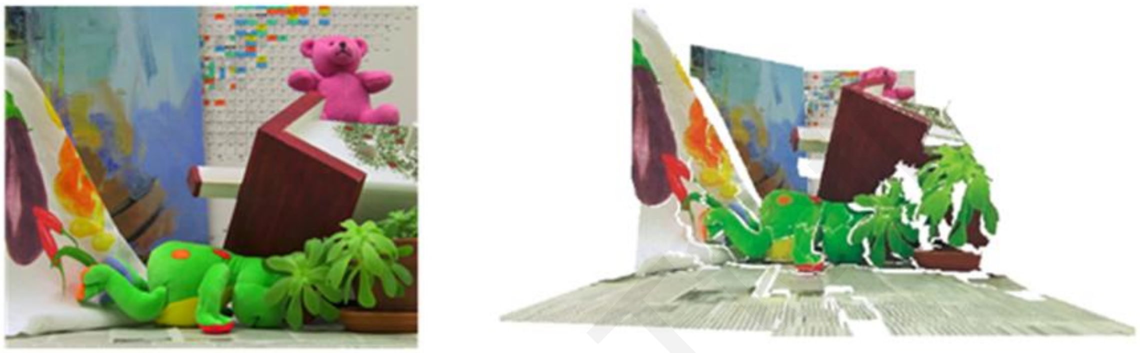

(a) (b)

**Figure 2.5: (a) Teddy (left view). (b) Pseudo 3D depth map representation** [**24**].

### 2.1.3.3 *Pseudo-3D representation*

This representation aims at visualizing the reconstructed scene from different points of view. Figure 2.5 provides such an example for the Teddy image [**24**].

## 2.1.4 Standard Testbeds

The use of standard testbeds and datasets has always been essential for the quantitative evaluation of computer vision algorithms. Several areas of computer vision including object recognition and stereo rely on challenging datasets to perform objective comparisons, track the progress made by leading algorithms and to stimulate new ideas. One of the most important testbed used in depth estimation, and more specifically in stereo vision, is the Middlebury database and evaluation testbed [**25**], [**26**]. This database includes a set of benchmark stereo images along with known ground truth disparity maps (as will be shown in subsequent sections, the disparity map is simply the reciprocal of the depth map) that allow the quantitative evaluation of depth estimation algorithms using the following quality measures:

1) The root-mean-square (RMS) error in *(2.1)* measured in disparity units between the computed disparity map $d_C(x,y)$ and the ground truth disparity map $d_C(x,y)$, where $N$ is the total number of pixels in the image.

$$RMS = \sqrt{\frac{1}{N}\sum_{(x,y)}(|d_C(x,y)| - |d_T(x,y)|^2)} \quad (2.1)$$

2) The percentage of bad matching pixels *(2.2)*, where $\delta_d$ is the disparity error tolerance.

$$B = \frac{1}{N}\sum_{(x,y)}(|d_C(x,y)| - |d_T(x,y)| < \delta_d) \quad (2.2)$$

Both metrics measure the error in depth estimated values over the whole image, but it is also possible to compute these statistics on three different kinds of regions, namely *NON_OCC* (all points except for occluded areas), *ALL* (all points including half-occluded regions), and *DISC* (only points along depth discontinuities). The aforementioned regions are used to support the analysis of depth estimation algorithms in typical problem areas. Figure 2.6 illustrates a sample of four stereo pairs in the Middlebury database (from top to bottom: Tsukuba, Venus, Teddy and Cones) along with their corresponding ground truth disparity maps and the three kinds of regions just described.

It is worth noting that the statistics in equations *(2.1)* and *(2.2)* can also be computed for each different region independently (e.g. $RMS_{ALL}$, $B_{ALL}$, $RMS_{DISC}$, $B_{DISC}$, $etc.$) using the generic equations *(2.3)* - *(2.4)*, where the subscript $R$ identifies the set of points (region) over

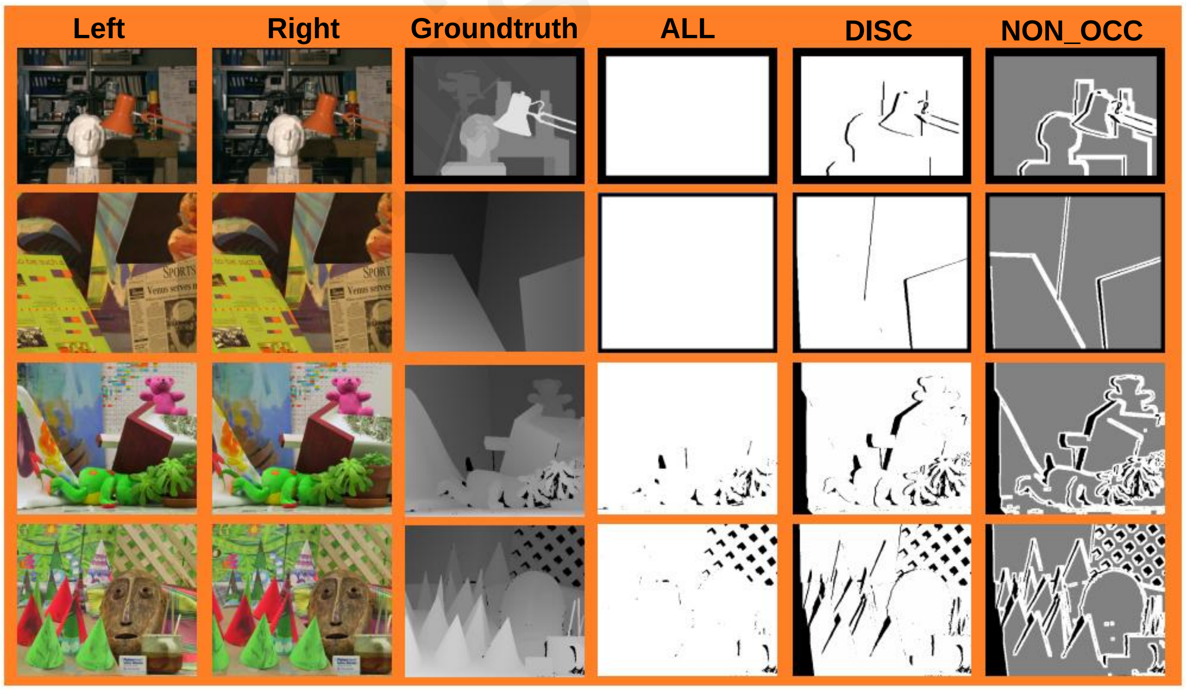


**Figure 2.6: Sample of stereo pairs in the Middlebury dataset with their corresponding ground truth disparity maps and the ALL, DISC and NON_OCC regions.**

which a statistic is measured ( i.e. $R \in \{NON\_OCC, ALL, DISC\}$ ), $d_C(x,y)$ and $d_T(x,y)$ are the computed and ground truth disparity maps, respectively, $N_R$ is the total number of pixels in the region of interest, and as already mentioned before, $\delta_d$ is the disparity error tolerance.

$$RMS_R = \sqrt{\frac{1}{N_R} \sum_{(x,y)\in R} (|d_C(x,y)| - |d_T(x,y)|^2)}, \tag{2.3}$$

$$B_R = \frac{1}{N_R} \sum_{(x,y)\in R} (|d_C(x,y)| - |d_T(x,y)| < \delta_d) \tag{2.4}$$

Since its development in 2003, Middlebury has been established as the de-facto testbed for the evaluation of depth estimation algorithms. However, the popularity of the depth estimation problem in research has recently led to the development of other testbeds as well. The .enpeda image sequence analysis test site (EISATS) [**27**] offers sets of image sequences for the purpose of comparative performance evaluation of stereo vision. The provided image sequences include both synthesized as well as real-world scenes. Another testbed that provides challenging real-world stereo vision benchmarks is the KITTI vision benchmark suite [**28**]. Recent research providing experimental results using the last two benchmarks indicates that algorithms ranking high on the Middlebury testbed, perform below average when being moved outside the laboratory to the real-world scenes [**29**]. Therefore, these benchmarks aim primarily in reducing the bias to a single benchmark, and complement existing benchmarks by providing real-world benchmarks with novel difficulties to the computer vision and depth estimation community.

## 2.2 Depth Estimation Technologies

The problem of depth estimation has been intensively investigated over the last decade. Various depth estimation techniques have been technologically demonstrated through a number of “depth cameras” or “depth sensors”, which integrate specific hardware capable of retrieving depth information about a scene either using a particular type of sensor (e.g. laser sensor ) or by running an algorithm on the images obtained by general camera sensors (stereo cameras). Depth sensors that can acquire a continuous stream of depth images are now commonly available and used in several applications, such as robotics and automotives [**30**], [**31**]. However, the different depth sensors differ considerably both in terms of their working principle, and also in terms of their resolution and precision. Therefore, choosing a particular depth sensor over the others is not always a trivial task, and it is strongly dependent on the targeted application and its requirements. This section provides a brief overview of the working principle / technical specifications of each depth camera, along with positive and negative aspects of each of them, and examples of products available in the market. Finally, a discussion on whether a particular depth camera is more suitable for specific applications will be made.

### 2.2.1 Time-of-flight

Time-of-fight (ToF) depth estimation cameras use the known speed of light to measure the time an emitted pulse of light takes to arrive to an image sensor [**32**], [**33**]. These sensors can be used to produce a depth image directly, without the use of traditional computer vision algorithms. The principle of ToF depth sensing is illustrated in Figure 2.7. A ToF camera consist of an IR emitter and an IR sensor, which emits IR waves to target objects, and measures the phase delay of reflected IR waves at each sensor pixel, respectively. The phase shift between the radiated and reflected IR waves is proportional to the distances from the reflecting surfaces, and thus allows the calculation of distances to objects.

ToF depth cameras provide accurate depth maps especially in highly dynamic environments, but they suffer from many systematic errors that are directed related to the sensor (e.g. noise and ambiguity), as well as non-systematic errors, such as scattering and motion blur, which are more strongly related to the scene-content [**32**]. Also, due to the periodicity of the cosine-shaped modulation signal, ToF cameras have a non-ambiguous range within which the distances can be computed uniquely [**34**]. Furthermore, according to the color, reflectivity and geometric structure of the target object, the reflected IR light shows amplitude and phase variations [**32**]. This causes depth errors in the resulting depth image. Last but not least, ToF are active sensors, and the amount of IR is limited by the power

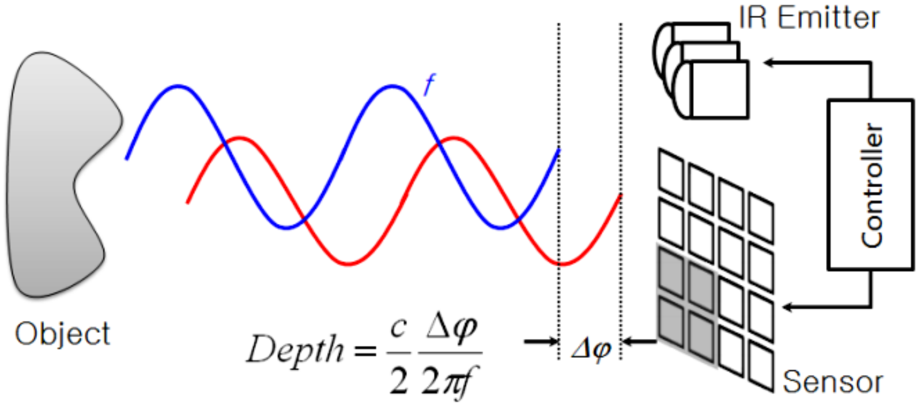


**Figure 2.7: Principle of operation of ToF** [32].

consumption of the device. As such, the reflected IR suffers from low signal-to-noise ratio (SNR). One way to increase SNR is to combine multiple pixels to calculate a single depth pixel; however, this reduces the effective image resolution.

An example of a ToF camera available in the market is the PMD[vision]® CamCube [**35**]. This camera has several limitations: besides being very expensive, it has a very limited frame size (200200 pixels) and a maximum frame rate of 40 fps. Moreover, due to the technology employed, the depth measurements can be very noisy and affected by the ambient light, limiting its use to indoor scenarios with controlled lighting.

### 2.2.2 Structured-light

The basic principle of structured-light depth cameras is the emission of a predefined light pattern (e.g. gray codes, sine waves, or speckle patterns), which is projected onto an object and simultaneously observed/captured by a camera (e.g. IR camera for IR patterns). Speckle patterns are used in popular structured light infrared-based cameras like the Microsoft Kinect [**36**] and the Asus XtionPro [**37**] sensors. These sensors construct the depth map by analyzing a speckle pattern of infrared laser light. It is worth nothing that the Kinect combines structured lights with classic depth cues adopted from computer vision techniques (depth from focus and depth from stereo). The image processor of the Kinect extracts depth from the deformation of the pattern; it uses the relative positions of the dots in the pattern to calculate the depth displacement at each pixel position in the image [**19**].

Structured-light depth sensors provide an attractive off-the-shelf solution to perform depth estimation with good accuracy and limited noise. Available depth cameras belonging to this category come at a low price (e.g. Kinect for Windows costs around $250), and also the depth computation is done entirely on hardware built in the depth camera, thus leaving the computational resources of the host machine intact. However, such sensors suffer from interference problems coming from nearby sensors, glass, water or sunlight. In such scenarios, the depth map information is affected considerably. In addition, these sensors usually support a limited operating range (0.8 - 3.5 m). These limitations make these sensors mostly suitable for indoor environments.

### 2.2.3 Stereo vision

The methods presented so far belong to a specific category called *active* depth estimation methods, as they work by putting some energy in the scene, projecting it in order to illuminate the space, and processing the reflected energy in a passive manner. On the other hand, stereo vision belongs to those depth estimation techniques that work with natural light in the ambient and the optical information of the capture images to estimate depth information [**21**]. These techniques use camera sensors to capture 2D images of the scene, and solve the depth estimation problem in a computational way. In other words, stereo vision is a *passive* depth estimation technique, focusing mostly on algorithms. Therefore, stereo vision sensors present a main advantage over the active ones, in that they do not transmit extract energy.

Stereo vision is inherent in humans and many other animals. Actually, it is what gives us the ability to see objects with height, width, and depth. In other words, we humans can see in 3D because we have two eyes. Each of our eyes sees the world from a slightly different perspective and our brain combines these perspectives to give us a sense of how close or far an object is. Computational stereo vision aims to recreate the human biological stereo vision and provides 3D perception by means of two spatially separated cameras that capture the same scene from different viewpoints. Due to the distance between the two cameras, an object in the real scene appears in each image with a displacement or disparity, which is inversely proportional to the depth. The problem of depth estimation can therefore be solved computationally by finding corresponding points in the two images (i.e. projections of the same 3D point), a process, which as we will see shortly, is known as stereo matching. Computer stereo vision is therefore much like human stereo vision, except that the medium by which 3D information is inferred is a computational platform, rather than the brain of some living creature.

When stereo vision technology was first introduced, it had the disadvantage of requiring more than one camera sensor and a powerful host machine to execute multiple computationally expensive algorithms. However, with the availability of very powerful and low-cost host machines (processors, DSP, FPGA, etc) and the reduced cost of CMOS camera sensors, this technology is finally established and widely used. Another reason that leaded to their wide adaptation and made them a valuable solution to the depth estimation problem, is their unique properties in augmenting the scene reconstruction with highly detailed depth and color data in areas and conditions where ToF and structured-light cameras may not provide sufficient details [**38**], [**39**], [**40**]. A widely known stereo camera that is used in research projects and industrial application is the Bumblebee camera [**16**]. This depth camera supports a resolution of 640x480 at 30fps or a maximum resolution of 1280x960 pixels at 15 fps. It also supports a very large depth (16-bits depth pixel) with a working range of 0.5–4.5m. However, the camera is currently quite expensive (around $2000), and the entire depth calculation is performed entirely on a host computer, which makes the total cost higher. It is anticipated that the proliferation of low-cost stereo vision cameras that can interface with low-power computational platforms such as FPGAs, will enable the adaptation of stereo vision technology in a wide variety of new markets and applications. The last chapter of the thesis deals with an example of such a system implemented on a low-end FPGA board connected to an inexpensive stereo camera, which is used in the application of obstacle detection for autonomous navigation.

### 2.2.4 Conclusion

The different depth estimation technologies analyzed above have advantages and disadvantages, regarding energy needs, computational load, accuracy, range, hardware implementation or price, among others. In general, it is not a trivial task to choose a winner among all technologies, as the different depth sensors work better in particular conditions and environments. For example, ToF sensors provide accurate depth maps especially in highly dynamic environments. They also work well on areas of the scene that have no texture (e.g. a wall painted in one color). However, they are currently limited by their low resolution and high cost [**34**]. Stereo sensors do not work well on texture-less regions, but perform better than ToF on textured scenes. Structured-light sensors provide an accurate and a cost-effective solution for depth estimation, but they have limitations in certain environments due to interference. Thus, they are tailored mostly for indoor applications. Stereo cameras on the other hand, are more expensive than structured-light sensors, and the depth computation is performed on the host machine, in a sequence of multiple computationally expensive steps. Hence, depth estimation through stereo vision relies on the design of efficient algorithms and architectures to enable real-time depth computation. Furthermore, the quality of the computed depth data in stereo vision systems is highly influenced by the scene illumination and surface texture. In general, stereo vision technology is associated with several challenges, which are extensively discussed in Section 2.6. Conclusively, one needs to consider the final application and the overall requirements, in order to make the correct choice. Furthermore, considering the complementary characteristics of the depth sensors, combining the depth information provided by different sensors could perform better than either sensor alone [**41**].

## 2.3 The Human Visual Perception of Depth

The human visual system interprets depth using a combination of different physiological and psychological cues. Physiological cues can be binocular or monocular, depending whether they require both eyes to be open, or if they are available even when looking at the world only with one eye open. On the other hand, all physiological cues are monocular. The main depth cues of the human visual system are briefly described below:

- ***Stereo disparity:*** humans have two eyes that see two slightly different images. The two images obviously provide an increased field of view, but also allow us to gauge depth and allow objects to appear in 3D. This is attributed to the difference in the sensed images called binocular parallax, or stereo disparity. The human visual system is very sensitive to these differences, which provide the most important source of depth perception in the human visual system; the sense of depth can be achieved using binocular parallax even if all other depth cues are removed.
- ***Accomodation*** is the information our brain receives from the muscles that changes the lens of eye and brings into focus objects at different distances (lens is made thinner as we focus on distant objects and thicker as we focus on near objects), as illustrated in Figure 2.8 (a). Accomodation is considered a binocular cue as it happens with both eyes, but it is still a monocular cue, as one eye alone gives the same information as the two eyes provide. Furthermore, this depth cue is generally weak, as it is efficient only at viewing distance less than 2 meters.
- ***Convergence*** is the difference in the direction of the eyes. As we focus on close objects, our eyes point slightly inward (Figure 2.8 (b)). As with accomodation, this depth cue is effective on short distances (less than 10 meters).

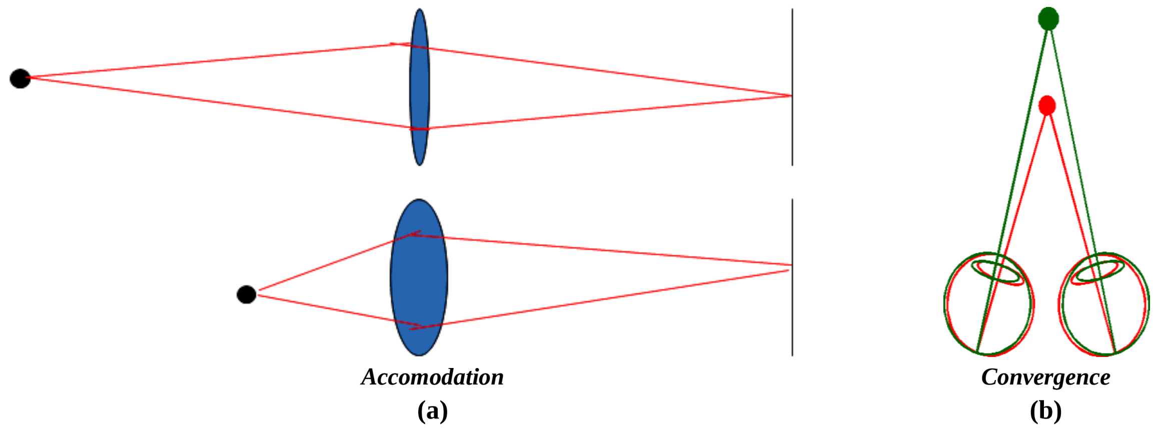


**Figure 2.8: Accomodation and Convergence in human visual perception system.**

- ***Interposition*** is a monocular depth cue derived from the overlapping position of objects [**42**]. Nearer objects may partially block our view of the more distant objects. Since the human brain is trained about the object structures, it interprets obstructed objects as being farther away.
- ***Texture gradient*** is a monocular cue that relates to how detail or texture the brain can sense according to our proximity to an object [**42**]. As the surface of an object gets farther away, the texture gets finer and appears smoother. It is this correlation between proximity and texture that is interpreted by the brain as a distance cue.
- ***Relative height:*** objects that are closer to the horizon seem more distant.
- ***Relative size:*** When objects are supposed to be the same size, then smaller objects are further away than closer ones.
- ***Linear perspective*** is the monocular cue provided by the convergence of lines toward a single point of the horizon.
- ***Relative motion parallax:*** Nearby objects move faster than more distant objects.
- ***Motion perspective:*** When moving towards an object, more centered parts seem to move slower than parts on the edges

In summary, the human visual system is able to recover depth information even from a single still image by relying only on monocular cues in the image. However, while depth perception from single images is a problem that humans solve fairly well, this problem is indeed impossible in a narrow mathematical sense, and traditionally is considered impossible for machines and computers. Consequently, machines and computers mostly rely on binocular cues and stereo vision to infer depth information of objects in a scene.

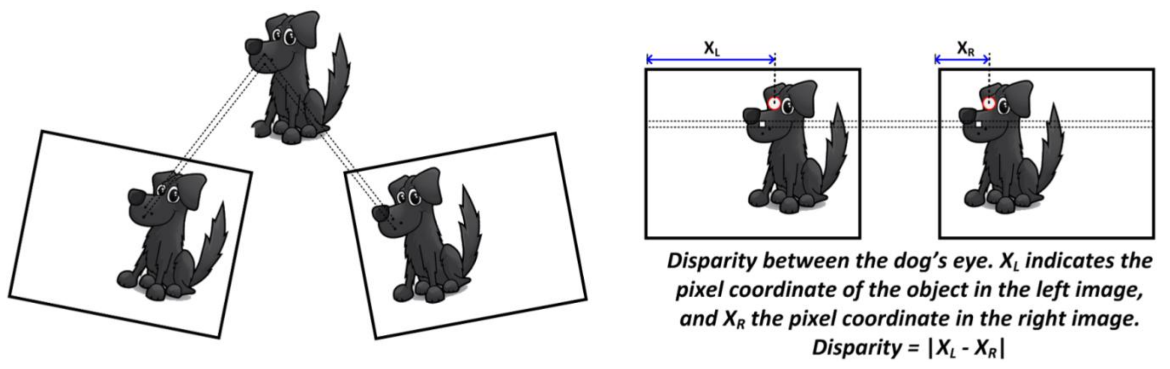
## 2.4 Depth Estimation using Stereo Vision

Vision is undoubtedly the most important sense of humans that provides them with the ability to understand the surrounding world and perceive the depth of objects in the scene. This great ability enables humans to accomplish many routine daily activities such as determining how far away is a car from other objects, climbing a flight of stairs, etc. Therefore, endowing computers and other machines with vision, or the ability to see in the same way we can see, has attracted increased attention in the fields of Computer and Machine Vision for decades. Classical Computer Vision started by analyzing and interpreting the 2D image frames taken by a single camera. While this method provides relevant information on the surrounding world (e.g. edge detection, segmentation, object detection and classification), it lacks 3D vision, a certain important aspect from human vision that enables the inference of 3D information in the scene.

As discussed in the previous section, recovering 3D information from a single still image is a problem that humans solve fairly well relying only on monocular cues in the image. However, this problem is indeed impossible in a narrow mathematical sense, and traditionally is considered impossible for machines and computers. Computer vision systems utilize a greater number of cameras in the scene, and rely on binocular cues and stereo vision to infer depth information [**9**], [**10**], [**11**]. Since the researchers in Computer and Machine Vision fields aim to mimic the human vision behavior and functionality, a two-camera system, called stereo vision system, is usually preferred.

The operation of a machine/computer stereo vision system is based on the biological model of stereovision itself [**11**], where the distance between the two eyes is exploited to estimated the depth. Due to this distance, an object in the real scene appears in the vision obtained from each eye with a displacement or disparity as illustrated in Figure 2.9. This displacement is inversely proportional to the distance between the two eyes and the object itself [**11**]. Human brain is responsible for uniting the two separate images captured from each eye into one picture, by matching up the similarities in the two images and finding their differences. Every object appears in the combined three-dimensional stereo image as solid ("stereos" in Greek) in three spatial dimensions: width, height and depth. It is the added perception of the depth dimension that makes stereo vision so common and important in computer vision systems [**11**].

Following the biological vision system, a computer stereo vision system provides 3D perception by means of two cameras that capture the same scene from two different viewpoints. Depth information is inferred by searching and locating corresponding points in



*Disparity between the dog's eye. $X_L$ indicates the pixel coordinate of the object in the left image, and $X_R$ the pixel coordinate in the right image. Disparity = $|X_L - X_R|$*

**Figure 2.9: Stereo vision disparity.**

the two images. In general, video processing in stereo vision has to cover several stages until inferring the final distance measurements: from image acquisition and camera system modeling, to stereo matching, depth calculation and filtering. Next, we provide a short overview of stereo vision fundamental concepts, such epipolar geometry and camera calibration. We also describe the major steps involved in the stereo vision processing pipeline, focusing primarily on the matching processing itself, which is the major focus of this thesis dissertation.

### 2.4.1 Pinhole Camera Model, Perspective Projection and Epipolar Geometry

This section defines the concepts of pinhole camera model, perspective projection and epipolar geometry, which are necessary to understand the geometrical model of stereo camera sensors and the relationship between the sensors and the acquired images. The first step in a stereo vision system is to acquire a pair of 2D images of the surrounding world that can be further processed and used to reconstruct the 3D geometry of the scene. This means that image coordinates from the 2D space are utilized to infer the 3D coordinates of the objects in the scene. The ideal relationship between the 3D coordinates outside the camera and their corresponding image coordinates is found through the pinhole camera model.

The history of pinhole cameras goes as far back as the 4th century BC and the Greek philosopher Aristotle and Mathematician Euclid, who understood and wrote about pinhole cameras. A pinhole camera is a simple camera with an exceedingly small aperture and no lenses to focus light – effectively a light-proof box with a small hole in one side [**43**]. When a source of light shines on an object, each visible point of the scene reflects a ray of light that passes through this single hole, and projects an inverted image on the opposite side of the box. The concept of a pinhole camera is illustrated in Figure 2.10.

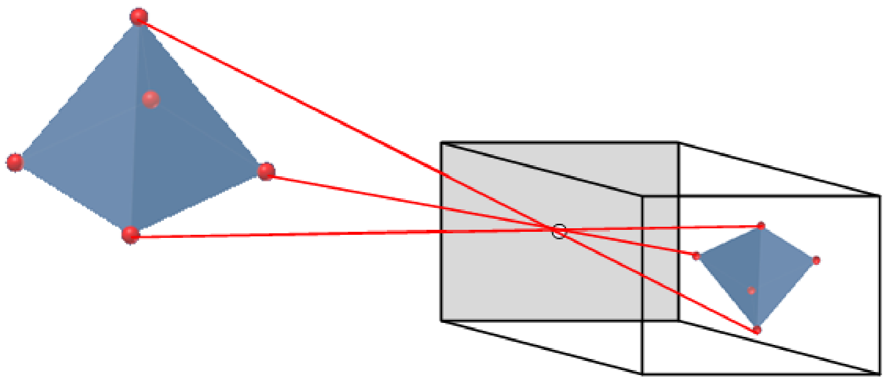


**Figure 2.10: The principle of a pinhole camera.**

Pinhole cameras describe the basic mathematical model that relates the coordinates of points in the 3D space and their corresponding projections onto 2D image planes [**43**]. The geometric mapping from 3D to 2D coordinates that follows the pinhole camera model is called a *perspective projection*. The center of the perspective projection is the point in which all the rays intersect, and is called the *optical center* or *camera center*, *O*. The plane over which the inverted image is projected is called *retinal plane* or *image plane, R*. A pinhole camera is completely modeled by its optical center and the retinal plane. The distance from the principal point to the retinal plane is referred to as the *focal length*, *f*. The *optical axis* is the line perpendicular to the image plane, passing through the optical center, while the intersection point of the image plane with the optical axis is called the *principal point*. The pinhole camera model and perspective projection, along with the fundamental definitions presented in this section are illustrated in Figure 2.11.

The pinhole camera model, while simple, is sometimes used to model real cameras. A real camera has a lens that directs the path of light rays to a film or a sensor array to recreate the image. The camera lenses have optical distortions [**44**], but since these can be removed by calibration and rectification procedures (discussed later in this section), modeling real cameras with the pinhole camera model is a valid assumption.

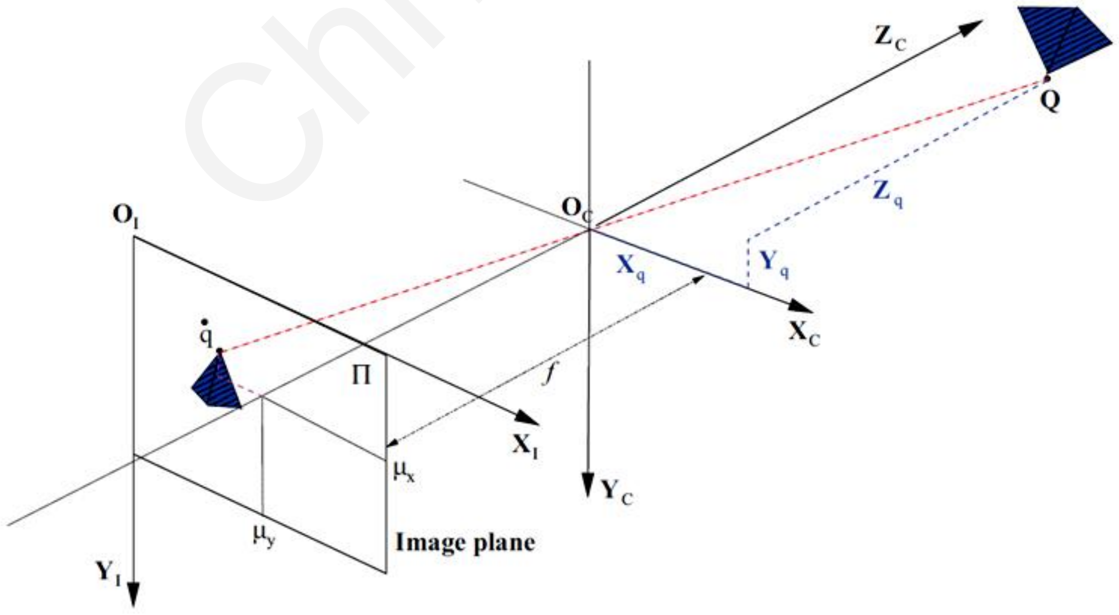
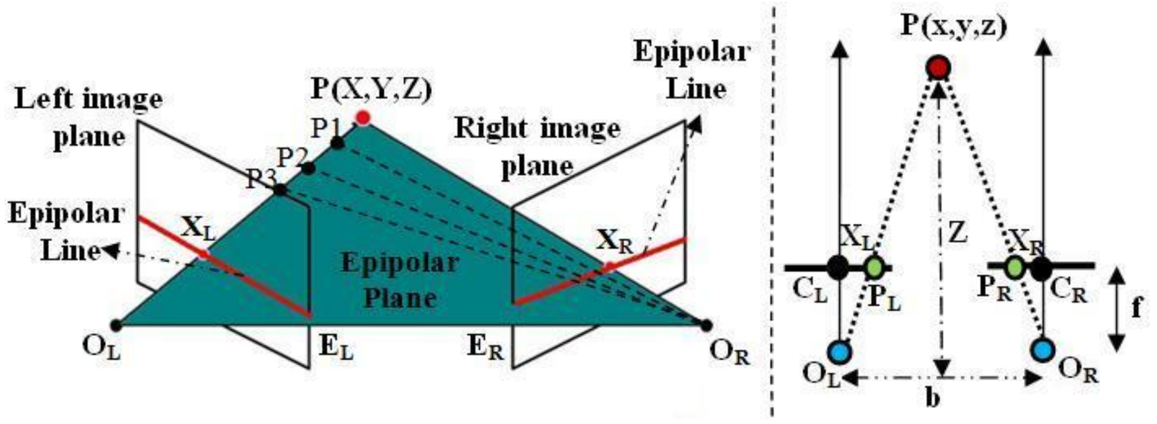


**Figure 2.11: The perspective projection.**



3D view 2D view

**Figure 2.12: The epipolar geometry.**

Following the pinhole camera model, only simple geometry and algebra is needed in order to understand how 3D points can be located in the 3D space using a stereo camera system. Figure 2.12 shows two pinhole cameras and their optical centers, $O_L$ and $O_R$. *P(x,y,z)* is a point in 3D space, $P_L$ and $P_R$ are the projections of *P(x,y,z)* in the left and right images, respectively, and the two points, denoted $E_L$ and $E_R$, represent the *epipoles*. An e*pipole* is the point of intersection of the line across the optical centers, i.e. the baseline, with the image plane, while the points *P*, $O_L$ and $O_R$ form a plane called the *epipolar plane*. The line $P$-$O_L$ is seen by the left camera as a point $P_L$ because it is directly in line with the camera's center of projection $O_L$. In the right image plane, that line is represented as the line $E_R$-$P_R$ and is called the *epipolar line*. For each point observed in one image, the same point must be observed in the corresponding epipolar line on the other image. This is known as the *epipolar constraint*, which establishes a mapping between points in the left image and epipolar lines in the right image, and vice versa. The search for correspondences is thus reduced to a 1D problem (searching along conjugate epipolar lines) [**9**]. Given pixel coordinates of $P_L(x_L,y_L)$ in the left image, and the corresponding pixel coordinates $P_R(x_R,y_R)$ in the right image, the 3D world coordinates *P(x,y,z)* are computed as in (2.5), where *b* (baseline) is the distance between the centers of projection $O_L$ and $O_R$. $x_L$ and $x_R$ are the coordinates of $P_L$ and $P_R$ with respect of the principal points $C_L$ and $C_R$. *f* is the common focal length and *z* is the distance between *P* and the baseline *b*, which represents the depth of the object. The *disparity*, *d*, is defined as the distance of two corresponding points, and is computed by $d = x_R - x_L$ (Figure 2.12). Disparity represents the relative difference in the location between common objects (i.e. the location of the object in the right image, relative to the location of that object in the left image) on the stereo image pair. The disparities of all common points form a disparity map, from which information about the depth of objects can be extracted using (2.5); the depth map is obtained by the reciprocal of the disparity map.

$$x = \frac{bx_L}{d}, \qquad y = \frac{by_L}{d}, \qquad z = \frac{bf}{d} \tag{2.5}$$

### 2.4.2 Major Steps in a Stereo Vision System

An overview of a stereo vision system is shown in Figure 2.13. It receives a pair of stereo images (left and right image) as an input, and outputs a disparity map (or, the depth map). A stereo vision system must solve two basic problems: *correspondence*, which deals with finding an object in the left image that corresponds to an object in the right image, and *reconstruction*, which deals with finding the depth (i.e., the distance from the cameras that capture the stereo images) and structure of the corresponding object of interest. The correspondence problem (also called matching) is the most demanding in terms of computational complexity, and involves searching and matching techniques (to locate a common object in both images), the robustness of which determines the quality and precision of the reconstructed 3D data [**25**].

A common approach to locate a corresponding point in the two images lies in using small windows (called *search windows* or *correlation windows*) from both images, which are being evaluated by comparing the window from the left image to a sliding window from the right image. The search for stereo correspondences can be constrained along a horizontal scanline, as the images are rectified. Additionally, an object that appears in both stereo images will be found within a maximum horizontal bound, which depends on the object's minimum distance from the camera. Henceforth, a search limit can be imposed, known as *disparity range* ($[d_m, d_M]$), which constraints the search along a bounded horizontal scanline. The size of

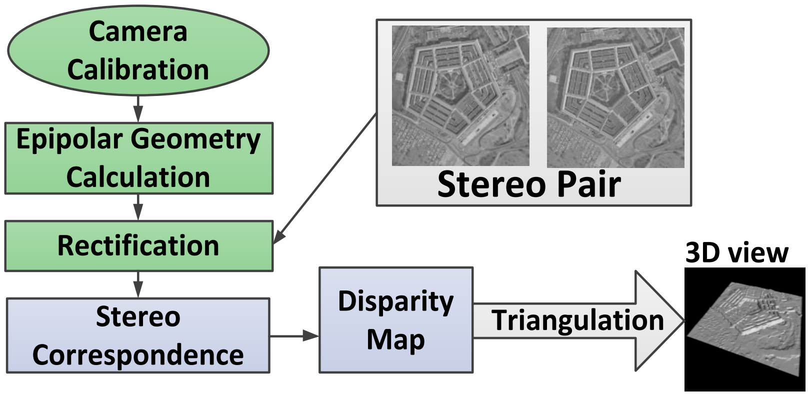


**Figure 2.13: Stereo vision system overview.**

the search window, and the disparity range, impact the reconstruction algorithm significantly, both in terms of performance, as well as quality of the results. When an object is located in both images, its pixel coordinates are then used to derive the disparity map, and by using triangulation, the 3D structure of the scene. However, there is a plethora of other algorithms for locating corresponding points between stereo images. These are discussed in Section 2.5.

### 2.4.2.1 *Calibration of Stereo Cameras*

Calibration is a procedure usually performed offline with the aim of finding the geometry of pair of cameras. This geometry is fully described upon calibrating two kinds of parameters:

- *intrinsic* parameters that characterize the transformation mapping of image points from camera to pixel coordinates. These include the focal length and image center (called principal point) for each camera, as well as parameters of lenses distortion, etc.
- *extrinsic* parameters that describe the relative position and orientation of the two cameras, through rotation and translation matrices that align the two cameras; they bring the reference frame of the two cameras onto each other.

The most-often used calibration method is carried out by acquiring and processing more than 20 stereo image pairs (to obtain a solid calibration) of a known checkerboard pattern with a defined square size and number. The calibration procedure is described in detail in [**45**]. Calibration is available in both OpenCV [**46**] and MATLAB [**47**]. In this thesis, the Bouget's

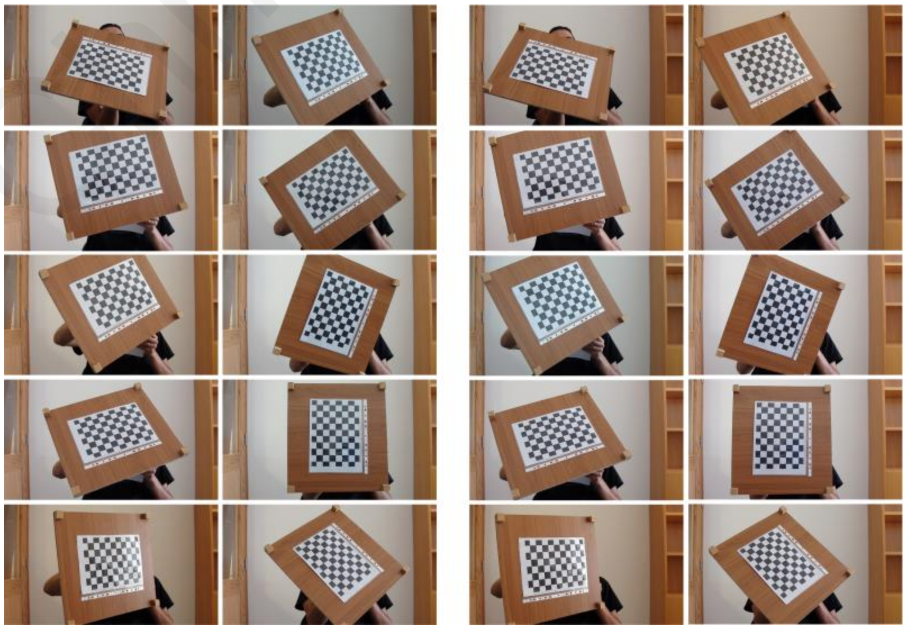


**Figure 2.14: Images used to calibrate the left camera.**

[**47**] and RADDOC [**48**] MATLAB toolboxes were used. The former toolbox requires interaction with the user in the form of corner picking, while the latter is an optimized version that utilizes Harris corner detection to automatically locate the checkerboard corners. Figure 2.14 shows 20 stereo pairs used to calibrate the stereo setup integrated with the proposed disparity estimation architectures.

Recent research in the area of stereo camera calibration has directed towards calibrating the cameras from the data available during use, rather than relying on data extracted online. This process, which is called self-calibration [**49**], makes the complete stereo vision process more flexible, but at the same time, adds extra computational needs.

### 2.4.2.2 *Stereo Image Rectification*

Equation (2.5) holds true when a pair of stereo images comes from calibrated cameras (with known focal length, and fixed baseline distance), and has been rectified. Rectification is the process that generates images conversion matrices based on characteristic values (intrinsic and extrinsic parameters) resulting from camera calibration, and apply these matrices (called *homography matrices*, *H*) to respective images acquired by the stereo camera, in order to transform each image into a common image plane, thus aligning the pairs of conjugate epipolar lines to a common image axis (usually the horizontal), as illustrated in Figure 2.15. The transformation from coordinates from the original images ($x$, $y$) to coordinates in the rectified images ($x''$, $y''$) is illustrated in equations (2.6)-(2.7). This transformation is usually implemented with reverse mapping and interpolation, in order to avoid problems such as reference duplication [**50**]. The transformation function can remain the same if the stereo camera system remains static and calibrated, otherwise, a new transformation function is necessary [**50**].

$$\begin{bmatrix} x_l' \\ y_l' \\ z_l' \end{bmatrix} = H_l^{-1} \begin{bmatrix} x_l'' \\ y_l'' \\ 1 \end{bmatrix} \quad , \quad \begin{bmatrix} x_r' \\ y_r' \\ z_r' \end{bmatrix} = H_r^{-1} \begin{bmatrix} x_r'' \\ y_r'' \\ 1 \end{bmatrix} \tag{2.6}$$

$$\begin{bmatrix} x_l \\ y_l \end{bmatrix} = \begin{bmatrix} \frac{x_l'}{z_l'} \\ \frac{y_l'}{z_l'} \end{bmatrix} \quad , \quad \begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \frac{x_r'}{z_r'} \\ \frac{y_r'}{z_r'} \end{bmatrix} \tag{2.7}$$

Rectification is an important step in a stereo vision system in order to enable real-time computation. While the epipolar constraint reduces the search for corresponding points to a single dimension (along the epipolar lines), rectification ensures that the computation of stereo

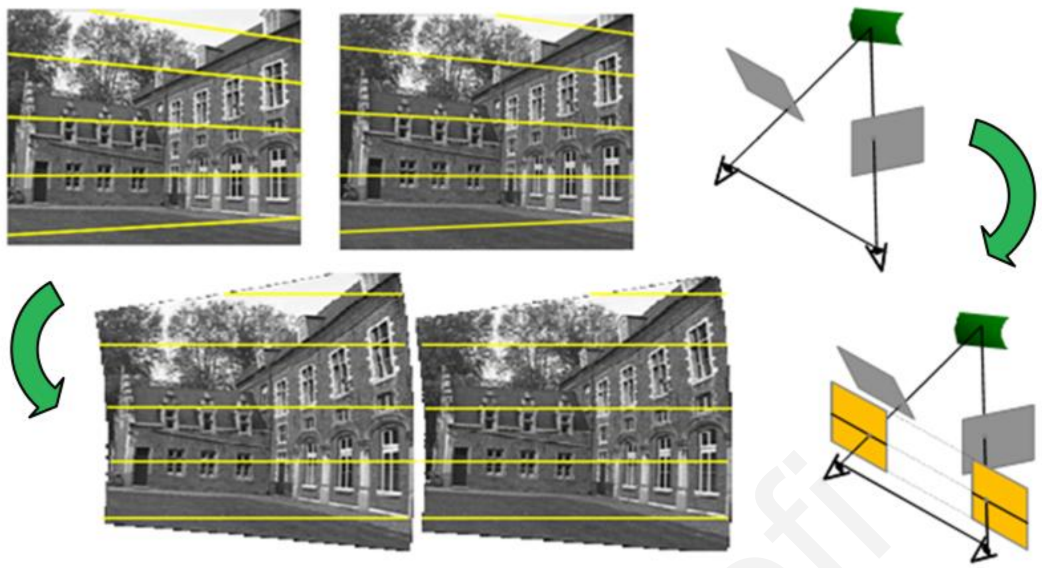


**Figure 2.15: Rectification transforms each image into a common image plane, aligning the pairs of conjugate epipolar lines to the horizontal image axis [51].**

correspondences is reduced to an 1D search problem along the horizontal raster lines of the rectified digital images, as the transformation function aligns the epipolar lines with the horizontal scanlines of the images [**50**]. By making the two images row-aligned, the rectification step allows easier access to pixel values and helps in avoiding a lot of computations that are required to keep track of the epipolar lines for the pixels in the reference image being searched in target image. Moreover, this row-alignment benefits hardware implementations of stereo vision systems, as the pixels can be processed in scanline order. This reduces memory usage, makes memory management easier and enables synchronization of stereo processing modules with the pixel clock of the stereo camera.

The transformation function in (2.6)-(2.7) implies that the rectification process involves complex operations, including multiplication and inversion of 3x3 matrices. Therefore, the implementation of the rectification step, either in software running of general purpose processors or in custom hardware, should be treated with care in order to enable real-time computation. In order to reduce the overall system complexity, many implementations [**52**], [**53**], [**54**], [**44**] pre-store the mapping, and use Look-up tables (LUTs) to perform the transformation; however, at the expense of consuming excessive memory for high resolution images and loosing flexibility when camera parameters change at a later time. One solution to the former issue is to use differentially encoded LUTs, an approach found to obtain a compression ratio of around 70% [**55**].

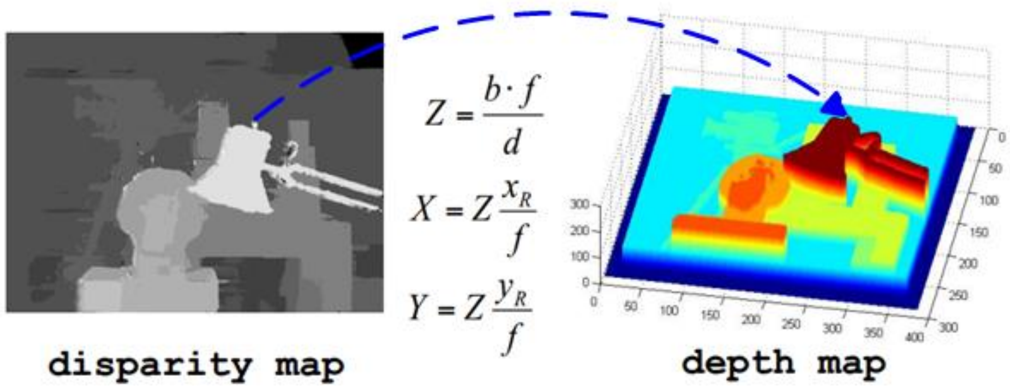


**Figure 2.16: Triangulation computes 3D coordinates of corresponding points.**

### 2.4.2.3 *Stereo Matching*

Stereo matching infers depth information from a pair of rectified stereo images (called *reference* and *target images*) by locating corresponding pixels between the reference image $I_r$ and the target image $I_t$. Given that the input images are rectified, the correspondence of a pixel at coordinate ($x$, $y$) in $I_r$ can only be found at the same vertical coordinate $y$ in $I_t$, and within a maximum horizontal bound, called *disparity range D* ($d_m$–$d_M$). The disparity is computed as the absolute difference between the coordinates of the corresponding pixels in $I_r$ and $I_t$. The disparities of all corresponding pixels form a disparity map. More details on the stereo matching process, including the major steps involved in this process and a classification of different stereo matching algorithms can be found in Section 2.5.

### 2.4.2.4 *Triangulation*

Given the baseline and the focal length from the calibration step, and the disparity map from the stereo matching step, triangulation computes the position of each correspondence in the 3D space (Figure 2.16). Triangulation is sometimes referred as 3D reconstruction, as it reconstructs the 3D point from its two projections (their distance $d$, more specifically).

## 2.5 Overview and Classification of Stereo Matching Algorithms

Stereo matching is a flourishing research area that has attracted the interest of many researchers so far. Active research in this field has resulted in a wide range of algorithms with primary targets to achieve real-time speed and/or improve the matching accuracy. The different algorithms are extensively studied in [**25**], which also provided a testbed for quantitative evaluation of the algorithms based on the *overall rate of bad matching pixels* relative to ground truth disparity maps of benchmark images (the widely know Middlebury testbed presented in Section 0). According to [**25**], stereo matching algorithms mostly follow four steps:

- *matching cost computation*, where a cost is computed for every pair of pixels;
- *cost aggregation*, where the initial matching costs are spatially aggregated either globally or over a local neighborhood;
- *disparity computation/optimization* that determines the best correspondence for each pixel;
- *disparity refinement*, where the mismatched pixels are removed and interpolated.

Some stereo matching algorithms also add a pre-processing step in order to alleviate sensor noise and photometric distortions. This is typically achieved by employing methods such as Laplacian of Gaussian (LoG) filtering [**56**], histogram equalization/matching, subtraction of mean values computed in the neighbors of each pixel [**57**], and bilateral filtering [**58**]. The rest of this section provides details for each of the major steps involved in the stereo matching process, and also provides a summary of the most important stereo matching algorithms.

## 2.5.1 Stereo Matching Four-Step Pipeline

### 2.5.1.1 *Matching Cost Computation*

The matching cost computation step assigns a cost value for every individual pixel at all possible disparities as shown in Figure 2.17. There is a plethora of cost measures that can be used to determine similarity between a pair of pixels. Existing cost metrics include absolute, squared and sampling-insensitive difference, or truncated versions, both on gray or color pixels. Some algorithms have also adopted a different approach to the matching cost computation step that makes uses of non-parametric local transforms (e.g. Rank/Census) as the

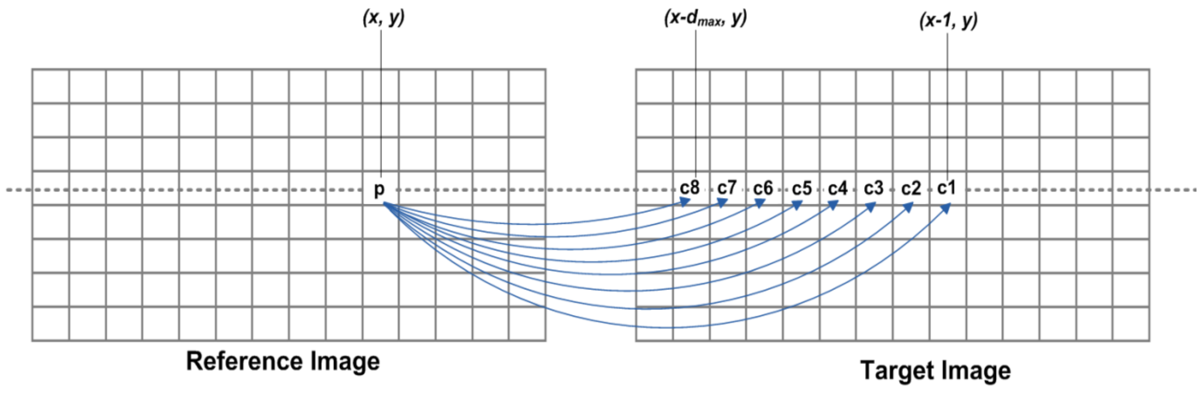


**Figure 2.17: The pixel-wise cost computation process.**

basis for matching. Non-parametric transforms rely on the relative ordering of local intensity values, and not on the intensity values themselves. Therefore, correlation using such kinds of transforms can tolerate a significant number of outliers, and also eliminate sensitivity to radiometric gain and bias [**59**]. The transformations are applied over the input stereo images, converting each pixel value into a string based on a set of comparisons in a local window. The matching costs are then computed using the hamming distance between corresponding strings.

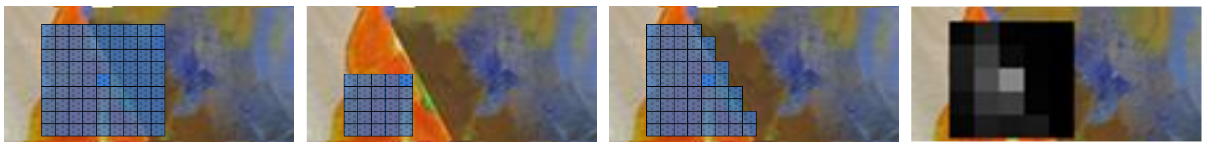
Absolute difference and hamming distance over Census transformed images are the most endeavored metrics exploited for real-time stereo matching in embedded vision systems, mainly due to computational simplicity and matching reliability. Moreover, recent approaches have suggested performing correlation based on combinations of the aforementioned metrics (e.g. absolute difference with gradient and census measures) to produce reliable matches.

#### 2.5.1.2 *Cost Aggregation*

Pixel-wise cost computation is generally an ambiguous process. It is possible that wrong matches have a lower score than correct ones, mainly due to noise from lighting variations and environmental conditions. To reduce matching ambiguity, the matching costs are aggregated by summing over a local *support region* rather than computed pixel-wise, in order to average the noise (reduce the signal to noise ratio) and therefore obtain more reliable estimates of the costs. Table 2.1 lists the most common cost aggregation methods. Note that each method is defined according to the pixel-wise cost metric used to measure the similarity between pairs of

**Table 2.1: Common Cost Aggregation Methods.**

| Cost Metric | Description | Definition |
|---|---|---|
| **Sum of Absolute Differences (SAD)** | Aggregates the color (luminance) difference of reference pixels and target candidate pixels. | $\sum_{u,v} \lvert I_r(u,v) - I_t(u+d,v) \rvert$ |
| **Sum of Square Differences (SSD)** | Square difference approach squares and aggregates the difference of reference pixels and target candidate pixels. | $\sum_{u,v} (I_r(u,v) - I_t(u+d,v))^2$ |
| **Normalized Cross-Correlation (NCC)** | Normalized Cross Correlation. The cross correlation is normalized by the mean value in the block. Higher NCC stands better match. | $\frac{\sum_{u,v}(I_r(u,v) - \bar{I}_r) \cdot (I_t(u+d,v) - \bar{I}_t)}{\sqrt{(I_r(u,v) - \bar{I}_r)^2 \cdot (I_t(u+d,v) - \bar{I}_t)^2}}$ |
| **Rank** | Rank transform calculates the number of neighbor pixels which have the value lager than the central pixel. The matching cost is calculated from the absolute difference of the two ranks. | $\sum_{u,v} (I_r^{'}(u,v) - I_t^{'}(u+d,v))$ <br> $I_{r,t}^{'}(u,v) = \sum_{m,n} I_{r,t}\,(m,n) < I_{r,t}(u,v)$ |
| **Census** | Census Transform encodes the comparison result of central pixel and neighbor pixels into a bit string. The matching cost is calculated from the hamming distance of census bit string of corresponding matching candidates. | $\sum_{u,v} HAMMING(I_r^{'}(u,v), I_t^{'}(u+d,v))$ <br> $I_{r,t}^{'}(u,v) = BITSTRING_{m,n}\big(I_{r,t}(m,n) < I_{r,t}(u,v)\big)$ |



**Fixed Support Aggregation** | **Varying Window Size and/or Offset Aggregation** | **Multiple-Window Aggregation** | **Adaptive Support Weight Aggregation**

**Figure 2.18: Cost Aggregation Strategies.**

pixels. Among the methods listed in the table, the most commonly used cost aggregation methods are the *normalized cross correlation (NCC)*, the *sum of square differences (SSD)*, and the *sum of absolute differences (SAD)* [**60**], which is the simplest in terms of computational efficiency, and therefore, perhaps the most compatible with embedded constraints. Aggregation methods that first apply a local transform (such as Rank or Census) to the input images rely perform aggregation based on the *sum of hamming distances (SHD)*.

Besides the type of the correlation metric, the *cost aggregation strategy*, which defines the type and size of the local support region, plays a rather significant role in the processing performance and matching accuracy of the stereo matching algorithm. One of the earliest and most simple cost aggregation strategies relies on rectangular windows with fixed weights only (fixed shape and size). Alternatives to this basic idea were proposed more recently and attempt to improve the matching accuracy mainly by following either of two different approaches. The first approach allows the use of different window sizes instead of a single window of constant size. The other assigns adaptive rather than fixed weights to the points belonging to the window. In this way, it enables the use of windows whose size and shape change adaptively from pixel to pixel. The different aggregations strategies along with their pros and cons are briefly described below, while they are also illustrated visually in Figure 2.18.

#### *2.5.1.2.1 Constant Window Aggregation (CW)*

A first category of cost aggregation methods relies on a set of usually square windows that are fixed in both size and shape. The two windows are symmetrically defined on the reference and target image to aggregate any similarity measure cost, *C*, at each possible disparity level, as in (2.8), where *N* is the set of pixels around the central pixel of the window. It is worth observing that in CW aggregation the support region depends entirely on the exact values of both $I_r$ and $I_t$, meaning that the weights assigned to the pixels in the windows are fixed. Therefore, CW is an efficient approach when considering computation complexity, as it can take advantage of techniques such as integral images and incremental calculations to make the aggregation process independent of the correlation window size. However, it assumes that every pixel in the window should share the same disparity. This assumption is violated in image regions in which the disparity is discontinuous (i.e. at object boundaries). CW also performs badly at regions such as boundary, slant surface, and repetitive pattern.

$$C_{CW}(x,y,d) = \sum_{\forall (x',y') \epsilon W_N(x,y)} C_{W_N}(x',y',d) \tag{2.8}$$

### 2.5.1.2.2 *Aggregation based on varying window size and/or offset (VW)*

This approach tries to improve the accuracy of the computed correspondences by finding an optimal support window for each pixel. This is achieved by evaluating the local variation of intensity and disparity, using a set of window pairs having different sizes and displacements, and selecting the window pair that minimizes the similarity function, as in (2.9). This method performs well along depth borders, since it aims at determining the most appropriate point to center the window (by finding the displacement) in order to aggregate points lying at the same disparity plane as the central pixel of the window. In addition, it attempts to select a proper window size to account for image regions with low texture. However, this aggregation scheme is highly dependent on the initial disparity estimation and also results in rectangular or constrained-shaped windows. Furthermore, it is associated with high computational complexity, mainly due to the larger number of window pairs that evaluated to select an optimal support region.

$$C_{VW}(x,y,d) = min\left(\sum_{\forall (x',y') \epsilon W_n(i,j)} C_{W_n}(x',y',d) \;:\; n\epsilon[N_{min}\ N_{max}]\right) \tag{2.9}$$
$$i\epsilon[x-n, x+n], j\epsilon[y-n, y+n]$$

### 2.5.1.2.3 *Multiple-Window Aggregation (MW)*

The previous aggregation schemes use one window pair representing the best support. An alternative scheme is to employ not only a single window pair, but a set of window pairs to represent the best support region. The multiple windows aggregation strategy [**61**] selects an optimal support region comprised of a subset of predefined multiple windows, which are located at different positions with the same shape. As a result, the support region is not only constrained to rectangular shapes, but it is made out of partially overlapping windows having uniform weights, something that is equivalent to the case of having a single rectangular window with each point of the window weighted differently [**61**]. However, the limitation of this approach is that the shape of the local support region is not general, thus, it is inappropriate for pixels near arbitrarily shaped depth discontinuities.

$$C_{MW}(x,y,d) = \sum_{\forall(x',y')\epsilon W_n(x,y,d)\cup\{W_n(x\pm n,y\pm n,d)\}} C_{W_n}(x',y',d) \ \ : \ n\epsilon[N_{min} \ \ N_{max}] \tag{2.10}$$

### 2.5.1.2.4 *Adaptive Support Weight Aggregation (ADSW)*

An important generalization to the idea of determining a set of rectangular windows of varying size to allow supports to have unconstrained shapes and sizes is to assign different and variable weights to the points surrounding a pair of fixed, rectangular windows on $I_r$ and $I_t$. This category of aggregation strategy is referred to as adaptive support weight (ADSW) aggregation, and is currently the most accurate among existing aggregation methods. The weights assigned to the pixels of the support windows are generated adaptively, usually based on the content of the stereo images, and specifically, the color/ proximity distances of every pixel in the window with respect to the central pixel [**62**], as illustrated in (2.11). An improved version of this approach incorporates information extracted from image segmentation [**63**] within the ADSW cost function, based on the assumption that pixels belonging to the same segment should lie at the same disparity plane. In this way, ADSW aggregation averages only “relevant” pixels, leading to very good quality at depth borders [**62**], [**63**]. Despite their high matching quality, ADSW aggregation strategies are not suitable for efficient implementations in embedded vision systems, as they involve a lot of hardware demanding operations (e.g. multiplication, square root and division operations with floating point numbers). Moreover, unlike simple fixed support algorithms, it cannot take advantage of the “sliding window” technique [**64**], since the adaptive weights have to be recomputed at every pixel. Thus, cost aggregation needs to be performed in an exhaustive manner, making the hardware complexity directly dependent on the support window size; and to achieve good quality results in ADSW methods, large window sizes should be used [**65**].

$$\frac{\sum_{p_i\epsilon W_r,q_i\epsilon W_t,(x',y')\epsilon W_N(x,y)} w'_r(p_c,p_i)\cdot w'_t(q_c,q_i)\cdot C_{W_N}(x',y',d)}{\sum_{p_i\epsilon W_r,q_i\epsilon W_t} w'_r(p_c,p_i)\cdot w'_t(q_c,q_i)},$$

$$where \qquad w'_{r,t} = exp\left(-\frac{d_p(p_i,p_c)}{\gamma_p} - \frac{d_c\left(I_{r,t}(p_i),I_{r,t}(p_c)\right)}{\gamma_c}\right) \tag{2.11}$$

#### *2.5.1.2.5 Variants of Adaptive Support Weight Aggregation*

In order to accelerate the adaptive support weight cost aggregation process, many variants of the original ADSW algorithm in [**62**] have been proposed. These include algorithms that generate adaptive weights based only on the content of the reference image (asymmetric aggregation). Other approaches tried to speed up the aggregation process by generating weights and applying aggregation only on the vertical or the horizontal dimension around the current pixel (1D aggregation). In general, most variants of ADSW algorithms have adopted different shapes of support around the pixel being processed. The work in [**66**] uses a cross-shaped aggregation to derive a fairly compact representation of the support regions in an efficient manner. The key idea is that the cross-based representation of the support regions can be decomposed into two orthogonal 1-D aggregations, which can be implemented using an orthogonal integral image technique for fast cost aggregation over any arbitrarily shaped windows at constant time [**66**]. Other variants work by using smaller matching windows through down-sampling, rather than changing the shape of the support region. Finally, a reduced complexity adaptive support weight approach in [**67**] is based on an approximation of a bilateral filter [**68**], with the aim to make the complexity independent of the match window size. The aforementioned variants of ADSW aggregation strategies have shown increased processing speed compared to the original ADSW aggregation strategy in [**62**], however, at the cost of disparity map quality degradation.

### 2.5.1.3 *Disparity Computation/Optimization*

The cost aggregation strategies presented in the previous section were assumed to aggregate costs over local support regions. Algorithms that follow such aggregation strategies are referred to as *local* stereo algorithms. However, there are also algorithms that aggregate costs globally along the entire image. In the stereo matching literature, these algorithms are known as *global* stereo algorithms. These two different categories of algorithms result in different outputs in terms of computational complexity and matching accuracy.

The disparity computation /optimization step minimizes a predefined cost function either locally or globally, depending on the type of stereo matching algorithm implemented. In local stereo algorithms, a local optimization method called winner-takes-all (WTA) is used for disparity computation. Adopting a WTA strategy, the displacement or disparity (between the reference pixel and the matching pixel) that is associated with the minimum or maximum cost value, is selected at each pixel. On the other hand, global algorithms minimize the cost (or energy) function by solving an energy minimization problem, using techniques such as Dynamic Programming, Graph Cuts and Belief Propagation.

#### 2.5.1.4 *Disparity Refinement*

The disparity map generated by the disparity computation step usually contains ambiguous and inaccurate disparity values, which are caused by issues such as occlusions, camera noise, textureless regions and regions with repetitive patterns. Occluded areas refer to image points present in one image and not in the other, and for which there is insufficient information available to estimate a correct disparity. The other issues mentioned above lead to incorrect disparity computations as they cause the generation of multiple local minima. The different sources of errors in disparity estimation are described extensively in Section 2.6, which presents in detail the challenges of the stereo matching problem.

In order to eliminate or correct ambiguous/inaccurate disparities, a final disparity refinement step is often integrated in the stereo vision processing pipeline. This step performs the stereo matching one more time, by reserving the reference and target images and computing a second disparity. Assuming that the disparity map generated when using the left image as reference (left-to-right) is denoted by $D_L$, and its opposite disparity map (right-to-left) by $D_R$. Then, ambiguous/inaccurate disparities can be identified and marked as invalid if the disparity $x = D_L(i, j)$ and its corresponding disparity of $D_R(i + x, j)$ differ by more than a specified threshold. Optionally, invalid pixels can be interpolated based on the values of the nearest consistent disparities, or they can be left intact so that to enable higher level applications that use the disparity map as input to ignore these pixels, and rely on most confident disparity values instead.

Other procedures commonly used in the disparity refinement step include median filtering (unweighted and/or weighted) and sub-pixel estimation [**11**]. Median filtering performs noise reduction in the disparity map, based on its ability in removing "outliers" like salt-and-pepper noise, while sub-pixel estimation aims at increasing the resolution of the final disparity map.

### 2.5.2 Classification of stereo matching algorithms

As already mentioned in the previous section, stereo matching algorithms are classified into two broad categories: *global* and *local*. Local methods include techniques such as block matching, gradient-based optimization and feature matching, while global methods include techniques such as dynamic programming, intrinsic curves, graph cuts and belief propagation [**60**]. Global methods can produce very accurate results, but are slower and computationally more demanding compared to local methods, due to their iterative nature and high memory needs [**60**]. Local methods, on the other hand, are faster and less computationally expensive, hence suitable for the majority of real-time applications. Local methods can, at the same time, provide acceptable quality in the computed disparity map [**60**], especially methods that adopt complex cost aggregation strategies (e.g. ADSW). Since this thesis targets embedded vision applications with processing speed, power and hardware resource constraints, it focuses on local stereo matching algorithms. However, in this section we aim to provide a description of the most important stereo algorithms available in the literature (along with their pros and cons), because most of these algorithms are used for comparison purposes with the proposed stereo matching hardware designs. A summary of the different stereo algorithms described next is also given in Table 2.2.

#### 2.5.2.1 *Local Stereo Matching Algorithms*

Local algorithms generally form three broad categories: block matching, gradient-based optimization and feature matching. Block matching methods seek to locate a common point of interest in the two images by comparing a small window from both images using a similarity metric (Table 2.1). Early local algorithms relied on simple aggregation strategies that performed block matching by using either a fixed (typically square) window, or multiple windows with different sizes (window size varied based on a reliability measure) and/or displacements. However, these approaches are prone to matching errors at depth discontinuity regions; they blindly aggregate pixels belonging to different disparities due to the use of a fixed window (shape and/or size) [**60**]. The most recent local techniques attempt to improve the disparity estimation process at depth discontinuity regions by using an adaptive support window. These techniques are based on carefully designed cost aggregation strategies, and represent state-of-the-art methods in local stereo correspondence algorithms, since they can generate disparity maps that approach the accuracy of global algorithms [**60**]. These aggregation strategies include techniques such as adaptive support weight (ADSW) [**62**] and segmentation-based ADSW [**63**], and operate by assigning different weights to the pixels in a support window based on the proximity and color distances to the center pixel, or on information extracted from image segmentation. In this way, these methods aggregate only those neighboring pixels that are at the same disparity.

Besides block-matching, there are also two other broad categories of local stereo matching algorithms: gradient-based optimization and feature matching, which however are used less frequently compared to block-matching algorithms. Gradient-based methods seek to locate a common point of interest in the two images by formulating a differential equation relating motion and image brightness [**60**]. One problem with block matching and gradient methods however, is that both methods suffer from changes in pixel intensities, are sensitive to depth discontinuities and to regions of uniform texture in images [**60**]. Feature-based methods seek to overcome these problems by limiting the search space to specific, reliable features in the images, rather than pixel intensities themselves. Of course, this also limits the density of points for which a disparity value can be estimated, but this limitation can be avoided by using interpolation techniques, which fill in the points for which a disparity value is not determined by the stereo correspondence algorithm.

Feature-based methods first extract a set of potentially matchable image locations (called feature points), and use the detected feature points to highlight areas of high diversity in the image pair. Feature points may include edges, lines, regions, and gradient peaks. By only including feature points with high texture diversity, hard-to-match areas of low texture diversity are not matched. By correlating only feature points, the computational complexity is greatly reduced. However, a complete dense disparity map cannot be obtained. Additionally, feature-based methods have a great immunity to noise. Therefore, they may benefit applications where there is a need to match scenes with very different illuminations. In such cases, it is preferred to limit the output produced by stereo matching algorithms to stable features (e.g. edges) in the images, and therefore to matches with high certainty. It should be noted that the feature-based methods are intended to reduce processing time for stereo matching, so consideration is taken on choosing a feature extractor that does not consume more time than it saves.

#### 2.5.2.2 *Global Stereo Matching Algorithms*

Global algorithms formulate the stereo matching problem as an energy minimization problem, by imposing smoothness constraints on the disparities in the form of regularized Markov Random Field (MRF) based energy functions. However, optimization of MRF-based energy functions in an NP-hard problem, and as such, numerous approximation algorithms have been proposed, such as Dynamic Programming (DP), Belief Propagation (BP), Graph-Cuts, etc. A summary of the most known approximation algorithms for global energy

Table 2.2: Classification of stereo matching algorithms.

| Approach | Brief Description |
|---|---|
| **LOCAL METHODS** | |
| **Block Matching** | Finds the best disparity corresponding to the maximum or minimum cost of a cost function evaluated over a small region, using correlation-based or robust rank and census metrics. |
| **Gradient-Based Optimization** | Seeks to locate a common point of interest in the two images by formulating a differential equation relating motion and image brightness. |
| **Feature Matching** | Matches feature points rather than pixel intensities themselves. |
| **GLOBAL METHODS** | |
| **Dynamic Programming** | Finds the optimal matching disparity, by locating a 'path' through an image which provides the best (least-cost) match |
| **Intrinsic Curves** | Converts the search for the best disparity to a nearest-neighbors lookup problem, by mapping epipolar lines to intrinsic curve space. |
| **Graph Cuts** | Formulates stereo matching as an energy minimization problem on a weighted graph in which graph nodes represent image pixels, all possible discrete disparities are represented as a graph label set, and weighted edges between graph nodes correspond to defined energy terms. |
| **Nonlinear Diffusion** | Aggregates supports by applying a local diffusion process. |
| **Belief Propagation** | Determines the best disparity by passing messages (beliefs) between the pixels in an image. |
| **Correspondenceless Methods** | Deforms a model of the scene based on an objective function. |
| **OTHER METHODS** | |
| **Semi-Global-Matching** | Approximates a global 2D cost function using a sum of 1D optimizations from all directions through the image. |

minimization in stereo matching is provided in Table 2.2. Even though such approximation algorithms are capable to produce very accurate disparity maps, they still require large computational power and high memory capacities. For example, storing all messages (beliefs) in BP-based global stereo matching for a relatively small image pair of one Megapixel requires ~3 GB of RAM.

### 2.5.2.3 *Semi-Global Matching (SGM)*

An interesting alternative stereo matching algorithm that also attempts to achieve quality results that are on par with global stereo methods is Semi-Global Matching (SGM) [**69**], [**70**]. This approach can be considered in-between local and global methods. The basic idea is to renounce part of the accuracy by approximating a global 2D function using a sum of 1D optimizations from all directions through the image. SGM methods are therefore more affordable for dedicated hardware implementation, but they still consume excessive memory to store the temporary cost of different aggregation paths.

## 2.6 Challenges of Stereo Vision Systems

Stereo matching is from its nature an ill-posed problem with inherent ambiguities, which make it challenging to enable both accurate and fast disparity estimation results. The successful realization of a stereo vision system depends both on the chosen matching algorithm as well as on the computational platform used to implement the matching algorithm. To satisfy the requirements of applications where high matching accuracy is critical, the last few years have seen a resurgence of interest in the development of highly accurate stereo matching algorithms. Part of this interest has been spurred by fundamental breakthroughs in matching strategies and optimization algorithms, and part of the interest is due to the emergence of highly-parallel, powerful computing platforms. Unfortunately, implementations of complex disparity estimation algorithms usually struggle to achieve real-time processing on computational platforms suited for embedded vision applications, with limited hardware and power budget (the challenges related to the computing platforms will be discussed extensively in Section 2.8). Furthermore, while the most recent algorithms have improved the matching accuracy considerably, stereo matching still remains a difficult vision problem for the following reasons:

- **Camera system issues such as photometric variations, image blurring and sensor noise.** A stereo matching algorithm should be robust to such unavoidable issues (see Figure 2.19 a-b).
- **Missing information caused by occlusions, slanted surfaces, and other issues related to extracting information about three dimensions from two dimensional images [71].** Occlusions are regarded as one of the most challenging issues in stereo matching. These are pixels which do not really have a corresponding pixel in the other image (example shown in Figure 2.19 h). Therefore, there is insufficient information available to estimate a correct disparity for those pixels. Furthermore, when the matching process is performed along one direction only, occlusions can hardly be aware. However, in applications requiring dense disparity maps, occlusions need to be detected and handled appropriately. A good approach for handing occlusions in stereo matching is to treat the two images symmetrically and implement the stereo matching process in both directions. Occlusions can then be detected by finding where the two disparity maps are not negative of each other. When detected, occluded pixels can be marked as invalid so that not to be processed by the application, or they can be assigned a disparity value from their surrounding true

matches, wishing that this value is also a correct match for the occluded pixel. In the latter case however, this filling mechanism may fail in cases where a large area of occlusion happens intensively [**72**]. Additionally, computing the stereo matching process twice increases the overall computational complexity, especially in aggregation methods that are not symmetrical (e.g. adaptive weight methods that generate weights based only on the reference image).

- **Textureless regions.** These are areas where there is little or no texture in the scene (an example is shown in Figure 2.19 d. Such areas mainly constitute a problem in local stereo methods, which are dependent on the size of their correlation window. Local methods produce better results in textureless regions when using a large correlation window. However, some large textureless regions are still hard to handle. For example, if a textureless area is larger than the match window, a disparity estimate for a given pixel has not a unique match within a local region, leading to erroneous disparity estimates [**73**]. Global algorithms on the contrary use minimization of some global cost function that tends to improve disparity estimates in textureless regions. As already discussed however, the complex global optimization makes these algorithm slow for real-time applications.

- **Regions of repetitive structures and textures.** These regions (see Figure 2.19 e) are also regarded as challenging issues in local stereo matching methods. In such case, there are usually many “weak” minima of the matching cost (many aggregated matching costs close to the minimum matching cost), and small image sensor noise or lighting variations can easily lead to incorrect matches.

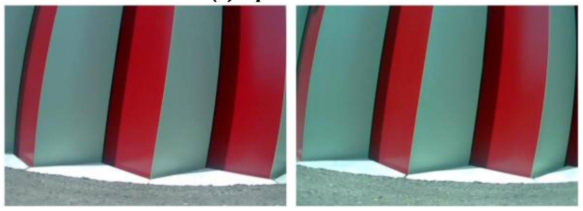
- **Specular reflections and transparency (non-Lambertian surfaces).** Stereo matching in the presence of specular reflections and transparency (see Figure 2.19 c, f, g) involves a significant difficulty, mainly because these effects violate the Lambertian assumption underlying traditional similarity-based stereo vision algorithms. Inferring correspondences by measuring similarity measures of potentially corresponding image pixels is strictly valid for surfaces with Lambertian (diffuse) reflectance characteristics [**74**]. However, in the presence of specular reflections (e.g. reflections from mirrors or a calm body of water) from non-Lambertian surfaces, the reflectance is viewpoint dependent. Therefore, objects in the real-scene may appear in the image captured by each camera with large intensity differences, leading to inaccurate correspondences, or no correspondences at all.

*(a) Photometric Variations*

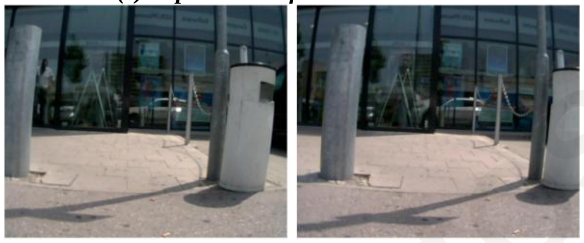
*(b) Image Sensor Noise*

*(c) Specularities*

*(d) Textureless regions*


*(e) Repetitive Surfaces and Texture*

*(f)Reflections*


*(g Transparency*

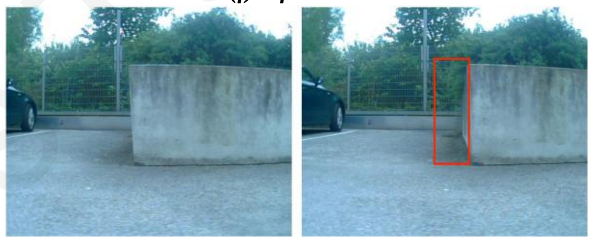

*(h)Occlusions*

**Figure 2.19: Stereo image pairs showing the challenges of stereo matching.**

- **Depth discontinuities.** Aggregation of matching costs over a local neighbor around the pixel being processed is done to average image noise. The use of small match windows leads to detailed but still noisy disparity maps. On the contrary, a larger match window results in smoother disparity maps, however, the problem is that it covers pixels which lie at different depths, thus leading to propagation of disparity information beyond the object boundaries.

- **Stereo camera calibration.** Calibration is a very important issue that must be addressed once for a given stereo camera system. Camera calibration determines the overall geometric structure of the camera model in relation to object distances; therefore it affects significantly the accuracy of the 3D measurements. Difficulties of perfectly calibrating/aligning the stereo camera system affect the epipolar constraint and result in misalignments in the stereo images [**75**].

- **Perspective distortions.** These refer to position and orientation differences of the 3D object relative to the position of the cameras. How a 3D object appears in each image plane depends on the camera (orientation, position and model), and the relative positions of the cameras with respect to the object. For example, an object of a particular shape (e.g. line, rectangle etc.) in one of the images may not correspond to the same shape in the other image, but to rotated or rescaled versions of the original shape. Therefore, the presence of significant perspective distortions leads to changes in object shapes that prevent features to correlate and result in unavoidable incorrect matches.

- **Perspective foreshortening**. Similar to perspective distortions, perspective foreshortening, an effect that occurs when a surface is viewed at a sharp angle, also leads to differences in the projection of an object in both images, reducing the accuracy of stereo methods.

## 2.7 Applications of Stereo Vision

As we discussed in Section 2.2, stereo vision is an attractive depth estimation technique that has the advantage of providing an accurate and detailed 3D representation of the environment, by passively sensing the environment and processing the captured images. Many products would benefit greatly from the ability to perceive depth information of the surrounding scene, in order to interact with their host environment and respond to events in real-time. Agricultural equipment, for example, could navigate fields autonomously, avoiding stray objects, people and wildlife. Cars could detect pedestrians and objects in the road, raising notifications to the driver or even break automatically. Security systems could track people moving through a building. Moreover, mobile robots with 3D vision capabilities could provide a clear visualization of critical infrastructures in crisis situations; the robots could be navigated in an autonomous way in the 3D environment to provide an overall situation awareness. It is imperative therefore, that stereo vision systems can provide accurate and real-time data in a variety of applications.

Point Grey [**76**], a company providing several stereo vision cameras, has also developed software that utilizes information extracted from stereo vision algorithms to detect and track the 3D positions of people in the camera's field-of-view [**77**]. By doing so, the company aims to enable several other applications in the fields of retail and security. For example, counting the number of people can be utilized in retail applications to improve store layout and evaluate the effectiveness of displays. Furthermore, the company mentions that the depth information provided by their stereo camera can also be used to improve customer service by recording queue length and average wait times [**77**]. Stereo vision can also benefit applications that need accurate three-dimensional, no-contact scanning of real-life objects, for example, medical imaging, 3D printing, etc. And plenty of other compelling applications exist such as 3D videoconferencing, manufacturing line "binning" and defect screening, etc.

Stereo vision technology is also becoming the preferred depth estimation technology in mobile robotics and automotive applications. This is attributed in a significant way in the recent advances in hardware computing platforms, which have given rise to the deployment of stereo vision systems able to produce dense disparity maps in real time. In addition, the fat that stereo vision is a passive technology that does not rely on beaming out laser or radar waves, gives it the advantage of not being susceptible to interference from other's robot/vehicle laser or radar beams. With stereo vision capability, a vehicle can determine not only that another vehicle or object is in the roadway ahead of or behind you, but also to accurately discern its distance from it. The following are a few examples of vehicles and robotic platforms that adopted stereo vision technology.

- The Princeton “Prowler” vehicle in the DARPA Urban Challenge [**78**], an autonomous vehicle research and development program that has robotic vehicles competing in a race in a complex urban environment, utilized the Bumblebee2 stereo camera for obstacle detection and navigation [**77**].
- Aldebaran [**79**] has recently developed a binocular camera system that is arranged horizontally and can be used to implement stereo vision algorithms of their NAO humanoid robot [**80**] (Figure 2.20 a).
- NASA’s Curiosity Mars rover [**81**], a car-sized robotic rover exploring the surface of Mars as part of NASA’s Mars Science Laboratory mission [**81**], used a stereo camera for navigation purposes (Figure 2.20 b). Not only in rovers, but stereo vision can be used for navigation of unmanned vehicles, submarines, etc. An example of using stereo vision technology for obstacle avoidance in autonomous robot navigation applications is presented in Chapter 6.2.
- Continental, an automotive supplier company, has recently announced a safety system for vehicles called ContiGuard [**82**]. Their safety system is actually a forward-looking braking system that uses stereo video cameras to identify pedestrians or crossing traffic

(a) (b) (c)

**Figure 2.20: Examples of real-world applications of stereo vision technology. (a) NAO humanoid robot from Aldebaran Robotics, (b) Mars rover, (c) Continental's ContiGuard forward-looking braking system**

when the driver does not do so fast enough, as depicted in Figure 2.20 c.

It is generally believed that stereo vision will be the technology of the future for depth perception, providing a cost effective alternative for applications which so far relied on expensive laser sources needed to illuminate the scene. The integration of stereo vision technology in many real-life applications is today a reality. However, there are many things to be done, in order to enable the integration of faster, more accurate and low-power stereo vision processing of high-definition images in a variety of exciting new products in embedded and mobile applications.

## 2.8 Parallel Architectures – Implementation Platforms

The previous section presented applications that can benefit from the ability of a stereo vision system to extract 3D information of objects in the scene in real time. In general, the amount of calculations a practical stereo vision system needs to perform is extremely high. Even the simplest local stereo algorithms need to evaluate at least $10^9$ disparities every second [**83**]. This amount of calculations is far beyond the capabilities of a single processor for real-time processing, leaving also little time for higher-level tasks. Given that stereo matching algorithms have high degree of inherent parallelism, their implementation using parallel architectures and state-of-the-art computing platforms has received considerable interest during the last decade. Three different architectures have dominated the field and used extensively towards the implementation of real-time stereo vision applications; *Multi-core Central Processing Units (CPUs)*, *Graphics Processing Units (GPUs)* and *Field Programmable Gate Arrays (FPGAs)*. This is attributed to their parallel capabilities that are usually needed when dealing with massive amounts of visual data. Another architecture that is worth noting for its parallel capabilities, although it has not been used so extensively, is the *Cell Broadband Engine (CBE)*. From the aspect of embedded stereo vision applications, these have substantially different constraints, such as low power consumption, limited hardware and memory resources, etc. Therefore, the architectures mentioned above are not all suitable for embedded environments, due to their high power consumption. In such cases, FPGAs, *Digital Signal Processors (DSPs)* and *Applications-Specific Integrated Circuits (ASICs)* are the preferred solutions. This section describes the different architectures and implementation platforms that have been used for the development of real-time stereo vision systems. The architectures are described in terms of development history, advantages and disadvantages (in terms of processing speed, power consumption, memory and communication overheads, etc.).

### 2.8.1 Multi-core Central Processing Units (CPUs)

Recently, CPU technology has seen a shift from trying to maximize the performance of a single core towards integrating multiple simple, low-frequency and low-complexity cores on a single chip. The latest processors available in the market include many physical CPU cores working together and communicate through the shared memory programming model. Furthermore, they also combine the Simultaneous Multithreading (SMT) capability. With SMT, the pipeline is duplicated to support multiple instruction flows (threads) in parallel, therefore increasing the total number of available cores through the "virtual" cores. Nowadays, SMT-based multi-core CPUs have become the trend in mainstream CPU technology, enabling the design of parallel applications, which can benefit from the extra computing power of more cores (physical and virtual). Computer vision applications are no exception. Several computer vision applications require heavy computation and lots of bandwidth in order to run in real-time, therefore they can greatly benefit from the ability of recent CPUs to run multi-threaded applications on multi-core processors. Other features of CPU technology that benefits vision applications include flexibility and ease of programmability (C/C++) and support for floating point operations. The programmer can easily develop parallel computer vision applications by using Application Programming Interfaces (APIs) like OpenMP that can split the work into multiple threads automatically. However, to develop scalable multi-threaded applications that best utilize the resources of emerging multi-core CPUs, the programmer needs to have a good understanding of the distinct characteristics of low/mid/high-level vision operations, and in some cases where real-time performance is critical, the computational workload of a vision application may need to be split into multiple threads in a manual fashion. From the perspective of flexibility, CPUs can be utilized to implement other applications that may not be inherently parallel more efficiently than GPUs and FPGAs, as they still offer the best performance in terms of single-core frequencies. On the contrary, they currently offer limited number of cores and consume excessive amount of power, factors that are prohibitive for highly parallel embedded vision applications.

### 2.8.2 Graphics Processing Units (GPUs)

A Graphics Processing Unit (GPU) is a specialized co-processor that can be found in nearly every computer system. It is traditionally used to enable acceleration of the rendering of 2D and 3D graphics by offloading it from the CPU. However, today’s GPUs have moved from being specialized rendering co-processors to more general parallel processors that are increasingly programmable to the point that they are capable of executing a significant number of computational kernels from many non-graphical applications. The fact that GPUs were initially designed to perform operations on a data structure (2D image) that is ideal for parallelization has allowed them to move to multiple processing cores much earlier than CPUs. Today, CPUs consist of a few cores optimized for sequential serial processing, while GPUs consist of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously. This Single Instruction Multiple Data (SIMD) capability makes GPUs a powerful platform for various computer vision algorithms. Although capable of providing significant performance gains through their streaming and data-parallel nature, they are highly power consuming and unsuitable to work as stand-alone processing units, as they are designed to work in tandem with the CPU. Therefore, GPU devices cannot currently find their place in embedded systems and mobile devices that typically have a quite tight power budget, particularly when they are powered by batteries. In cases where energy efficiency is an important design goal, embedded GPUs present a promising solution. However, OpenCL/CUDA drivers are currently available in the public realm only for a limited number of embedded GPUs. Furthermore, compared to high-end GPUs, embedded GPUS provide a restricted set of features, such as limited or no user-defined memory, small instruction-set, limited number of registers, among others.

### 2.8.3 Cell Broadband Engine

The CELL processor was built from Sony, Toshiba and IBM with the aim of being used in Sony’s PlayStation 3. Therefore, CELL is highly customized for gaming/graphics rendering. It implements a parallel architecture based on a SIMD computing architecture and high performance data transfer management, features that are of benefit for computationally intensive applications that contain inherent parallelism. The CELL architecture combines one powerPC multi-threaded core (PowerPC Processing Element - PPE) and eight specialized cores called Synergistic Processing Elements (SPEs). The different cores are connected with each other using a bus capable of transferring 25.6GB/s between each core connected to it. The PPE consists of the power processing unit (PPU), 512 KB 8-way write-back cache (L1) and 32 KB 2-way reload-on-error instruction cache (L2). The SPE includes a local store memory (LS), a memory flow controller (MFC) and a Synergistic eXecution Unit (SXU). The CELL communicates with the rest of the world, through its memory controllers and input/output controllers. Figure 2.21 (e) shows the architecture of the CELL processor. The real power of the CELL lies on the SPEs. Each SPE is a 128-bit RISC processor specialized for data-rich, computation-intensive SIMD applications. The SPE can only access its local store memory (LS), which is the main storage of each SPE. The exchange of data between the main memory of PPE and local memory of the other SPEs is realized using DMA. To sum up, the CELL processor is a suitable architecture for use with computationally demanding applications and algorithms that are inherently parallel. However, a major limiting factor of the CELL is the storage space of the SPEs; each SPE has 256 kb to store the data and also the code to process.

### 2.8.4 Digital Signal Processors (DSPs)

Digital Signal Processors (DSPs) appeared on the market in early 1980s, and since then, they have been used as the key enabling technology for many electronic products and accelerator systems in the fields of communications, multimedia, automotive, military, etc. A DSP is a specialized microprocessor optimized for the operational needs of real-time digital signal processing. Moreover, a DSP is designed to implement tasks in parallel (e.g. operations needed to implement an FIR filter can be completed in a single clock cycle). Since their appearance, DSPs have undergone an intense evolution in terms of hardware features, integration and software development tools. From a simple Harvard-based architecture supporting fixed-width instruction sets that included fixed-point addition, multiplication and accumulation operations, today's DSPs are based on parallel architectures, such as Very Long Instruction Word (VLIW) and Single Instruction Multiple Data (SIMD)) with many on-chip peripherals and added features such as pipelining, multiple dedicated arithmetic units, special address generation units, and Direct Memory Access (DMA) and support for floating point operations. They can easily be reprogrammed by software (usually C/C++), while their deterministic operations have known execution times, thus a DSP program can guarantee a desirable, repeatable performance. Additionally, DSPs enable high throughput that can sustain processing of high-speed streaming data (e.g. audio and image data processing), while at the same time they are optimized for low power consumption, therefore are well-suited for use in embedded environments.

### 2.8.5 Field Programmable Gate Arrays (FPGAs)

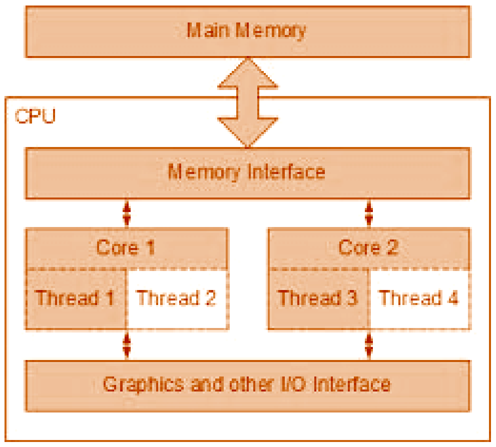
A Field Programmable Gate Array (FPGA) is a configurable hardware that can be programmed by the user with a circuit implementing a specific task. It consists of an array of programmable logic blocks, where everything runs concurrently. Therefore, FPGAs present an efficient solution to do parallel processing, and have already shown their high performance capacity in image and video processing applications [**84**].

FPGAs have almost zero non-recurring engineering (NRE) cost compared to custom ASICs. Design with FPGAs takes a few months to about a year; whereas the ASIC design cycle is 2-3 years. Therefore, FPGAs are more capable of serving the constantly changing requirements of the market; reduced time-to-market and design flexibility have made FPGAs relevant for more and more embedded applications. Designing with FPGAs requires easy-to-use and less complex design tools compared to ASIC design tools. High Level Synthesis tools have recently appeared as an increasingly popular approach to raise the abstraction level of digital design with FPGAs. However, optimized FPGA designs require deep knowledge of Hardware Description (HDL) languages, while coding is typically tailored to the specific FPGA device.

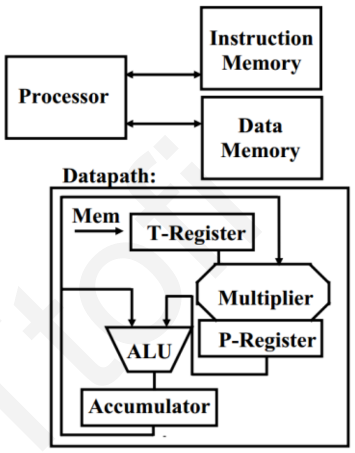
**Table 2.3: Comparison of the different implementation platforms.**

| Platform | Advantages | Disadvantages |
|---|---|---|
| **CPUs** | FP units, flexibility (C/C++) | power, size |
| **GPUs** | large number of stream processors, FP units | high power dissipation and cost, difficult to program |
| **Cell** | Flexibility (C/C++), deterministic performance | Limited resources, small size of directly accessible memory |
| **DSPs** | C/C++, low-power, low-cost | limited/shared resources, data word alignment and bandwidth issues |
| **FPGAs** | low power, high I/O capability, large scale parallel processing | coding is difficult (tailored for specific devices) |
| **ASICs** | lower unit costs, full custom capability, high density, power efficiency | long and complicated design cycle |

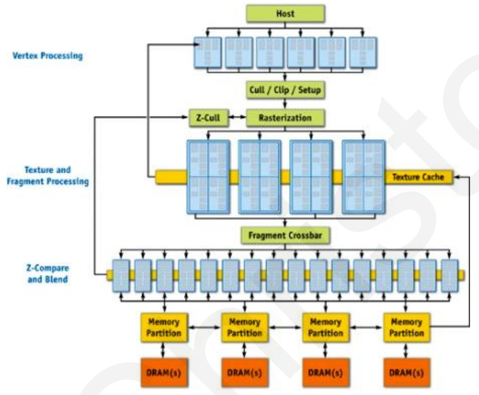
As the technology advances, FPGA density is increased; today, there are FPGAs with millions of logic cells, many integrated blocks and a large number of I/O pins in a very tiny area. Moreover, FPGAs include more features on chip, such as RAM blocks, DSP blocks, memory controllers, high-speed serial input/output transceiver, and hard Intellectual Property (IP) for PCIE, Ethernet MAC, and also Microprocessors. As new developments and



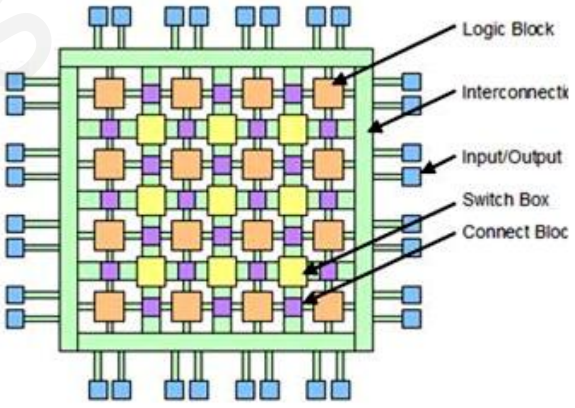
(a)



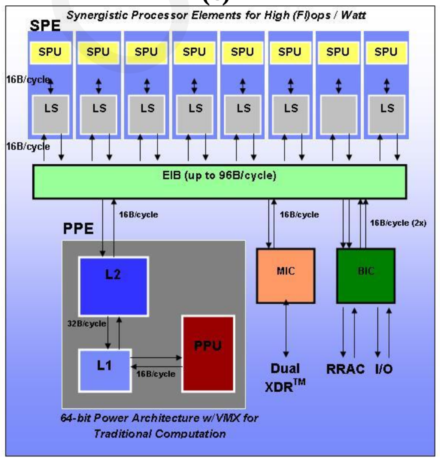
(b)



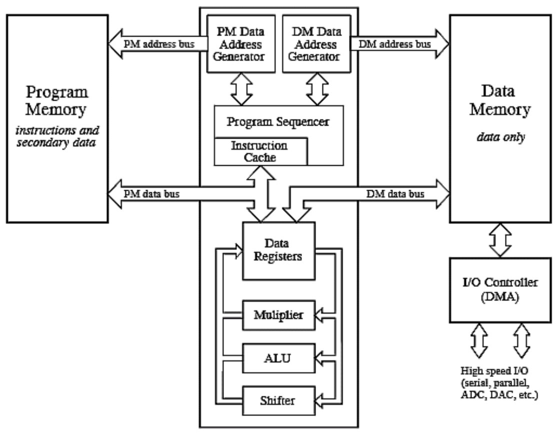
(c)



(d)



(e)



(f)

**Figure 2.21: Architectures of the different implementation platforms.**

innovations continue to happen to enhance FPGAs with additional flexibility, programmability and a wide range of new features, FPGAs would definitely be the preferred choice in applications that are complex and performance-intensive, especially when volumes are of the order of a few hundred per annum.

### 2.8.6 Application Specific Integrated Circuits (ASICs)

An Application Specific Integrated Circuit (ASIC) is an integrated circuit that is created to perform a specific function rather than to be flexible to do multiple functions. Due to this specialization, the circuit is able to perform the function was built at very high performance levels. ASICs have a higher R&D cost to design and implement, as compared to an FPGA. However, once an ASIC is fabricated, it is not reprogrammable like an FPGA is. The layout of the internal chip constructs are fixed and cannot be modified without a “re-spin” of the ASIC. This makes ASICs much less flexible than FPGAs. On the other hand, ASICs are more scalable than FPGAs in terms of useful logic, table sizes and resource availability, and thus are often used for very dense applications. Even though an ASIC may consume more power per unit die size than an FPGA, this power is amortized over a higher density solution; hence, an ASIC provides better power efficiency. Finally, while ASICs typically have a higher R&D cost to design, in high volume applications the lower costs of manufacturing ASICs are attractive.

## 2.9 Review of Existing Stereo Vision Implementations

As discussed in the previous section, stereo vision systems targeting embedded applications need to satisfy, often contradictory constraints, such as real-time processing speed, high disparity map accuracy and low power consumption. This makes the successful realization of an embedded stereo vision system a key challenge, both in terms of the chosen matching algorithm as well as in terms of the implementation platform. The current section shortly resumes the state-of-the-art stereo vision implementations considered for comparison purposes in next sections, highlighting their basic features (algorithm, implementation platform, etc.) and the performance metrics that are more desirable in each case. Comprehensive surveys of stereo vision implementations can be found in [**85**], [**86**].

### 2.9.1 Implementations based on General Purpose CPUs

While real-time disparity estimation can be achieved by software implementations running on general purpose processors [**87**], the accuracy of the disparity maps generated by these implementations is not very high, as they adopt simple local algorithms, therefore trading accuracy for speed. Software implementations of complex algorithms, such as global or local ADSW, generate very accurate disparity maps, but they rely on the high-end hardware resources of state-of-the-art processors to address the high algorithmic complexity and obtain real-time performance. An example of a real-time CPU implementation of a global algorithm based on dynamic programming (DP) is proposed in [**88**]. The work in [**89**] is another example that achieves real-time performance by making intensive use of the Streaming SIMD Extensions (SSE) and the multi-core architectures of state-of-the-art CPUs. However, due to the high computational complexity of the algorithms implemented in [**88**], [**89**], their iterative nature and high memory demands, the real-time performance of such implementations is limited to small-sized images (smaller than VGA). A recent SGM-based implementation [**90**] on a multi-core general purpose PC generates disparity maps whose accuracy is on par with global algorithms, such as DP and Belief Propagation (BP) algorithms. However, even with the use of parallelization and image sub-sampling, this implementation is able to compute 640x320 image pairs at ~14 fps, therefore achieving near real-time performance.

### 2.9.2 Implementations based on General Purpose Graphics Hardware

GPUs are attractive and capable platforms for the implementation of stereo correspondence algorithms, as evidenced in [**91**], [**92**], [**93**], [**94**]. GPUs have higher compute and memory bandwidth capabilities than CPUs, thus can support the algorithmic and memory complexity of accurate disparity estimation algorithms such ADSW combined with dynamic programming [**93**], global BP [**92**], [**94**] and SGM [**95**]. However, GPUs cannot be used alone, but work in tandem with CPUs, leading to complications with task decomposition and memory management [**96**]. Furthermore, they consume excessive power (in the order of hundreds of Watts [**97**]), which is not desirable in embedded applications. Hence, GPUs can only be a solution for PC-oriented applications of stereo matching, where power consumption is not an issue.

### 2.9.3 Implementations based on Digital Signal Processors

DSP platforms, on the contrary, feature lower power consumption than software implementations running on high-end processors or GPUs, thus they are more favorable in embedded stereo vision applications. One of the most famous, real-time DSP solution targeting robotic applications is presented in [**98**]. Another real-time DSP implementation of a local stereo matching algorithm based on jigsaw matching templates is presented in [**99**]. The works in [**100**] and [**101**] have proposed real-time and low-power embedded stereo vision systems that implement local algorithms on DSP processors. Recently, [**65**] have develop a fast stereo matching algorithm that was optimized for high-quality stereo engines for DSPs, achieving near real-time performance for 320x240 resolution images. The aforementioned works indicate that DSPs are capable to obtain very good processing performance on local, fixed-support algorithms. However, due to their limited resources, data word alignment and bandwidth issues, their computational power is not high enough to support real-time processing of complex disparity estimation algorithms. In such cases, real-time processing is obtained only for relatively small images and disparity range, which is not practical for several real-world applications, such as autonomous robot navigation.

### 2.9.4 Implementations based on the Cell Platform

Another solution for the implementation of stereo vision algorithms is the Cell processor. Recent works directed towards this approach indicate that the Cell processor platform can be used to parallelize complex stereo correspondence algorithms, such as ADSW [**102**], dynamic programming [**103**] and belief propagation [**104**], to yield very accurate results. The Cell architecture, however, is subjected to restrictions mainly due to the limited memory of the Synergistic Processing Elements (SPEs), and hence, the processing time obtained does not satisfy real-time performance.

### 2.9.5 Implementations based on Application-Specific Hardware Acceleration

Recent research indicates that application-specific hardware acceleration using either FPGAs or custom circuits (ASICs) might be the most appropriate solution for embedded stereo vision applications, since it can provide high computational power with low power consumption. Furthermore, it allows the architectures to be designed in a customized way; therefore, the computational resources can be optimized in terms of resource utilization. To this end, a lot of work has been carried out on real-time dedicated hardware implementations of disparity estimation algorithms on both FPGAs and ASICs. However, FPGAs remain the most popular implementation choice because of their inherent parallelism, re-programmability and great flexibility in manipulating the algorithm, and relatively short design cycle.

Dedicated hardware architectures suitable for real-time disparity map estimation are presented in [**105**], [**106**]. These works implement local, fixed support algorithms using the Sum of Absolute Difference (SAD) similarity measure, and compute intermediate-sized disparity maps at a rate of 768 and 600 fps, respectively. Another implementation of a more complex disparity algorithm based on locally weighted phase correlation is presented in [**107**], and utilizes 4 FPGAs to produce dense disparity maps of size 256x360 at 30 fps. A more recent FPGA implementation of a real-time stereo vision system is presented in [**108**], and generates dense disparity maps based on the Census transform. The hardware implementation presented in [**109**] performs a modified version of the Census transform in both the intensity and the gradient images, in combination with the SAD correlation metric (SAD-IGMCT algorithm), achieving 60 fps on 750x400 images.

The aforementioned dedicated hardware implementations indicate that the disparity map estimation can be effectively performed in real-time with high frame rates. However, many applications of disparity estimation require not only real-time processing but also reliable depth computation. Unfortunately, the majority of the aforementioned implementations struggle to provide reliable depth information, as they mostly implement fast fixed-support algorithms, which have difficulty in determining the best window size and shape for each pixel during the cost aggregation step. As such, these implementations are prone to errors and tend to generate incorrect matches especially at points along depth discontinuities [**62**]. The most recent dedicated hardware implementations attempt to alleviate the abovementioned problem by implementing modified versions of ADSW- and SGM-based stereo vision algorithms, thus enabling much better quality results.

FPGA implementations of SGM-based algorithms are presented in [**110**], [**111**], [**112**], [**113**]. The work in [**110**] introduces a SGM implementation based on a mean-free SAD (ZSAD) similarity criterion. This work achieves real-time throughput by sub-sampling the input images (by a factor of 2 in width and height) and reusing the result for full resolution computation, by parallelizing the path calculation step, and by combining 4 paths in one step to minimize the external memory bandwidth. The complete system matches two pairs of 320x200 images and a disparity range of 64 pixels at 27 fps; however, as stated in [**112**], that system has limited scalability to achieve higher data throughput. The work in [**112**] realized the SGM algorithm on a hybrid FPGA/RISC architecture, achieving 30 fps on 640x480 images at a 128 disparity range under 12MHz to 208MHz, depending on the amount of parallelism used. The architecture extends the initial implementation in [**111**] by introducing a novel 3D parallelization concept for the SGM to achieve higher throughput, but it currently supports processing of only 4 paths and integrates a simple post-processing pipeline (Left/Right check and 3x3 median filtering). Thus, optimizations are still needed in order to extend the architecture for higher accuracy. Moreover, the memory (path cost buffers) required to store the temporary cost of different paths increases with the number of paths and pixels processed in parallel, and also depends on the disparity range; thus, the architecture needs optimizations for high-resolution images. The FPGA implementation by [**113**] uses a memory efficient version of the SGM, called eSGM, in which the amount of temporary memory only depends on the number of pixels. This permits matching of larger images and reduces the requirements on the bandwidth. While the memory was reduced considerably, the eSGM exchibits 50% more compute operations as compared to SGM, and obtains lower accuracy.

ADSW-based dedicated hardware implementations follow the same concept with the SGM-based implementations. That is, a complex, but accurate, stereo matching algorithm is adapted for hardware implementation through a series of hardware-directed algorithmic modifications. To the best of our knowledge, there are currently two hardware implementations that are based on ADSW algorithms. The first one proposed in [**114**] presents a hardware-friendly disparity estimation algorithm called mini-census ADSW, and its corresponding real-time VLSI architecture. Their architecture achieves 42 fps on 352x288 image sizes, with very good accuracy. The other one proposed in [**115**], presents an implementation of a complete stereo vision system that incorporates an ADSW algorithm and integrates pre- and post- processing units. That system achieves 51 fps for 640x480 stereo images. However, there are still improvements that could be made, especially in terms of the accuracy of the matching cost function and post-processing integration, as well as in terms of the frame rate and scalability. It must be noted that the works in [**114**], [**115**], do not integrate segmentation information within the weight cost function, and also their aggregation methods are based on the reference image only. This does not permit the application of the mutual consistency check without doing the correlation twice.

The ADSW- and SGM-based hardware implementations described above have the potential to affect the trends in embedded stereo vision applications, as they are both based on stereo vision algorithms that achieve results that are on par with global algorithms. However, the local nature of ADSW algorithms and the reduced memory requirements compared to SGM algorithms have motivated us to implement an ADSW-based stereo vision algorithm. The proposed hardware implementation is based on an advanced segmentation-based cost aggregation strategy, aiming to maximize the speed-accuracy tradeoff. It must be noted that, between those implementations that an ADSW algorithm have been adopted, no one so far has deployed segmentation information.

## 2.10 Concluding Remarks

Stereo vision is an important component of human vision for depth estimation, and a key task in several embedded vision systems that require knowledge about the depth of objects in the scene. This chapter has reviewed fundamental theoretical aspects of depth estimation, placing particular emphasis on stereo vision technology and its basic theories such as calibration, rectification and stereo matching. Several stereo matching techniques were briefly introduced and classified into different categories. In addition, the chapter discussed the main challenges of stereo matching, and the various applications of this important process in the context of embedded systems. The chapter then provided an overview of the different processing devices and architectures that have the potential to exploit the parallelism inherent in stereo matching algorithms, and looked into existing stereo vision systems that have been implemented on these architectures.

# CHAPTER 3

# Edge-Directed Hardware Stereo Matching: Empowering Resource-Constrained Embedded Systems with Hard Real-Time Depth Computation

*THIS chapter presents an overview of the use of edge information as a means to accelerate hardware implementations of stereo correspondence algorithms. The presented approach restricts the stereo correspondence algorithm only to the edges of the input stereo images rather than to all image points, thus resulting in a considerable reduction of the search space. The benefits of the edge-directed approach are highlighted by applying it on a SAD-based fixed-support algorithm. The Chapter presents design considerations about the implementation of the edge-directed stereo algorithm on reconfigurable hardware and also discusses issues related to the memory structures needed, the amount of parallelism that can be exploited, the organization of the processing blocks, and so forth. The resulting edge-directed architecture is implemented on a Virtex-5 FPGA and is evaluated in terms of processing speed, disparity map accuracy, and hardware overheads, against a stand-alone SAD-based architecture and existing hardware stereo matching systems.*

## 3.1 Introduction - Motivation

The problem of enabling real-time disparity estimation in embedded vision applications has been investigated extensively during the last decade, leading to the development of numerous dedicated hardware systems. Common features found in most of these systems include the implementation of simple, mostly local, stereo matching algorithms, and the use of hardware mechanisms such as parallelism and pipelining in order to exploit the computation concurrency inherent in such algorithms. As presented in the literature review in Section 2.9, such implementations achieve satisfactory frame rates for relatively large image sizes and disparity ranges. However, due to the increased hardware complexity associated with larger image sizes and disparity ranges, satisfying the hard real-time constraints of emerging embedded stereo vision applications, especially for High Definition (HD) images and under limited resource usage and power budget, is still challenging.

With the aforementioned considerations in mind, this section presents the design of a hardware stereo matching system that incorporates edge information as a means to accelerate the overall stereo matching process and achieve real-time frame rates for HD images and under limited hardware/power budget. In particular, we present the hardware design of a block- and feature-based (hybrid) stereo matching algorithm; through an edge detector that generates the features (edges) used to reduce the search space, and a block matching Sum of Absolute Difference (SAD) algorithm used for the stereo matching computation. Alternative block matching algorithms can also be used as well; we chose the SAD matching technique, which is simple, fast and suitable for embedded applications. The integration of edge information constrains the stereo matching process only on binary data (edges), therefore reducing the search space and improving the overall frame-rate. Furthermore, edges represent reliable image features, and their use reduces the sensitivity to pixel intensity variations caused by camera gain or illumination changes. Therefore, the proposed edge-directed architecture outperforms traditional SAD block marching-based hardware architectures in terms of matching quality. Finally yet importantly, stand-alone SAD traditionally requires extensive resources, both in terms of memory and logic complexity. The edge detector implemented in the proposed hardware stereo matching architecture reduces the overall hardware requirements, making the SAD much more efficient than when used as a stand-alone matching technique. These features enable the design of a highly-parallel, scalable and resource-optimized architecture that is able to process HD stereo images in real time. Consequently, the developed architecture is particularly suitable for resource-constrained embedded vision systems that need to satisfy hard real-time and low-power constraints.

## 3.2 Edge-Directed Disparity Estimation System Overview

The proposed disparity map computation system follows the correlation window method (block matching SAD technique), combined with the features (edges) extracted by an edge detector, in order to reduce the data to be processed and to speed up the overall operation. The system also uses optimized memory access to the external memory, in an attempt to further increase the resulting frame rate, exploiting the fact that several computations involved in both edge detection and disparity computation receive as input overlapping pixel data. We assume

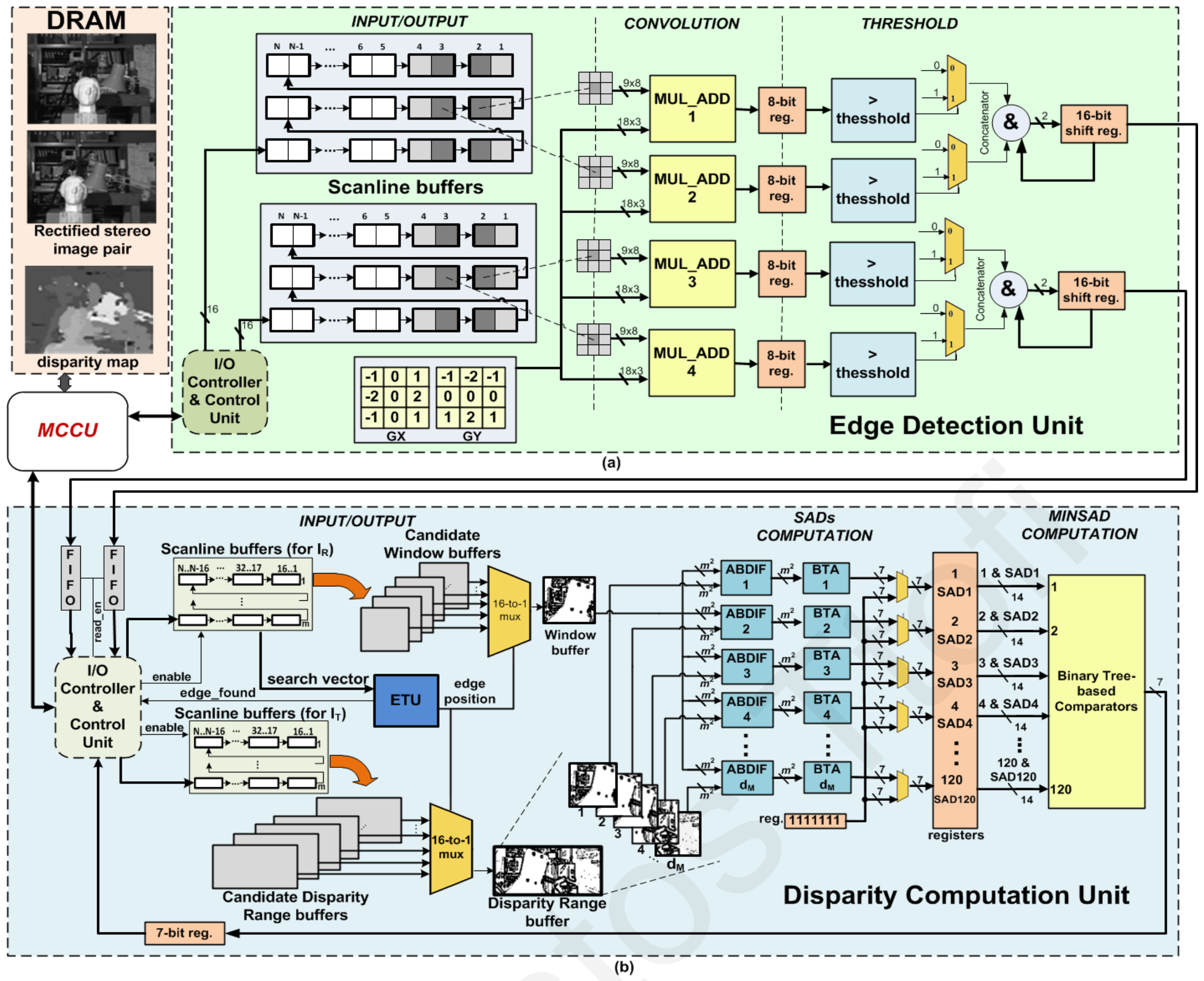


**Figure 3.1: Detailed block diagram of the proposed system architecture. (a) Edge detection unit (b) Disparity computation unit.**

that the input images are captured from calibrated stereo cameras [**10**], [**116**] and are rectified so that the epipolar lines become horizontal [**50**]- [**44**].

The block diagram of the proposed system architecture is shown in Figure 3.1. The system consists of two major hardware units: the Edge Detection Unit (EDU) and the Disparity Computation Unit (DCU). The system also consists of a Memory Controller & Control Unit (MCCU) that optimizes memory access based on the algorithm requirements, and coordinates data transfers and handshakes between the EDU and the DCU. The architecture is parametrizable in terms of the correlation window size, the image size and the disparity range. It receives a rectified stereo image pair as input, and produces a disparity map at the output. The EDU converts the rectified image pair (grayscale images) into a pair of binary images (black and white) that characterize only the edges of the initial images. The black and white images are then fed into the DCU, which performs correlation with the objective to compute the disparity map of the input image pair.

It is worth noting that while the disparity maps generated by the DCU are sparse (disparity estimates are provided only for points corresponding to edges), the proposed system can generate dense disparity maps as well, by integrating interpolation methods as part of the MCCU. In this work, we use the simple and fast nearest neighbor interpolation method; our emphasis has been on performance in embedded scenarios. We do however investigate the impact of more complex interpolation methods such as bilinear and bicubic interpolation.

The EDU and the DCU, which communicate through the use of internal memory (FIFO queues), are pipelined, and thus operate concurrently. They are also provided with scanline buffers, which temporarily store the pixels needed to perform convolution (in the case of the EDU), or correlation (in the case of the DCU). This reduces the clock cycles required to load image data from the input port, by exploiting the fact that working windows moved over the image use overlapping pixels. The scanline buffers are organized into FIFO structures and their size depends on the size of the working window (*3x3* for the EDU, *mxm* for the DCU) and the width of the image, *N*. The delay to fill the scanline buffers is proportional to the I/O bandwidth. A detailed description of the EDU and DCU is given next.

## 3.3 Edge Detection Unit (EDU)

Incorporating an edge detector into a stereo vision system can reduce the amount of data to be processed by the stereo correspondence algorithm. However, since correspondence will be established on a sparse set of image features (edges), the resulting disparity map will not provide disparity estimates for all image points. Thus, if the disparity map is to be used in applications that require dense disparity maps, interpolation methods will be necessary to fill in those points for which a disparity estimate is not determined by the stereo correspondence algorithm. This is not a problem, however, if the time saving due to the reduction of the search space is more than the overall delay of the edge detection and interpolation operations, which traditionally, are considered computationally less expensive than stereo correspondence algorithms [**117**].

As such, there are quite a few issues that need to be considered when selecting an edge detector for a stereo vision system, including the impact of the detector accuracy to the quality of the disparity maps, the computational delay the detector incurs, and the detector hardware efficiency. While there exist several edge detection methods, this work integrates the Sobel

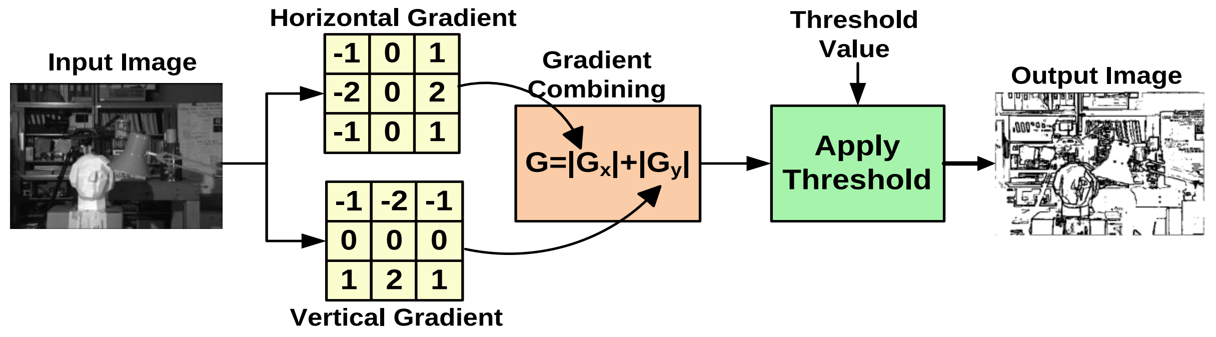


**Figure 3.2: Sobel edge detection overview.**

detector in the FPGA-based disparity computation system, mainly due to its simplicity and good performance on an FPGA [**118**]; detailed quantitative results for three different implementations of edge detection methods (Canny, Sobel and an algorithm proposed by Vasicek et. al. in [**119**]) are given in the Section 0, justifying our choice.

### 3.3.1 Sobel Edge Detector Overview

An overview of the Sobel edge detector algorithm is shown in Figure 3.2. The Sobel operator performs a 2D spatial gradient measurement on the input grayscale image using a pair of 3x3 convolution masks [**118**]. The masks hold data values between -2 and 2; thus the overall convolution can be implemented in hardware using shifters instead of multipliers. By avoiding the costly multiplication operation, higher frequencies are possible allowing integration of this method into disparity computation systems without affecting their performance. Moreover, the simplicity of the Sobel operator could allow for a parallel hardware implementation, capable of processing more than one image point at a time.

The Sobel edge detector is implemented in order to reduce the amount of data and speed up the overall operation. The black and white output image pair of the detector characterizes the feature boundaries (edges in black while non-edges in white). The matching algorithm matches windows whose central pixels represent edges. The fixed correlation window from the reference image is moved only to the edges (and not to each possible pixel) along the working scanline, resulting in a considerable reduction in the search space. Besides data reduction, the use of binary data can further speedup the overall operation, when compared to 8-bit operations.

### 3.3.2 Sobel Edge Detection Unit (EDU) Architecture

The proposed disparity computation system integrates a scalable and flexible Sobel Edge Detection Unit (EDU) shown in detail in Figure 3.1 (a). The EDU employs hardware features, such as parallelism and pipelining, in an effort to parallelize the repetitive calculations involved in the Sobel operation, and uses optimized memory structures in order to reduce the memory reading redundancy. The detector architecture consists of an I/O controller that reads/writes pixel data from/to the I/O port, on-chip memory for storing pixel data (scanline buffers) and the convolution mask values, a series of convolution units (MUL_ADD units), as well as comparators. The architecture is pipelined into 3 stages; *INPUT/OUTPUT*, *CONVOLUTION* and *THRESSHOLD*, a description of which is briefly given below:

•*INPUT/OUTPUT*: The I/O controller fetches 4 pixels from the I/O port (2 pixels from each image), in a row-wise fashion, and forwards them to the input ports of the scanline buffers (16-bits to each buffer). Each scanline buffer consists of a series of 16-bit registers (each register stores 2 pixels), which are organized into FIFO structures. When new pixel data is available at the input, the 16-bit registers are shifted one position to the right. Since each scanline buffer consists of registers, it can allow parallel access to its elements. Particularly, each scanline buffer produces two successive 3x3 windows. This pipeline stage is also responsible for writing the result of the overall operation to the output port; the I/O Controller asserts a signal indicating that the data at the output is valid.

•*CONVOLUTION*: This pipeline stage begins after the first 3 scanlines from both images are stored into the scanline buffers, and each buffer is able to produce 2 successive 3x3 windows per cycle. The four 3x3 windows (shown with gray color in Figure 3.1 (a)) are convoluted with the Sobel kernels using 4 MUL_ADD units.

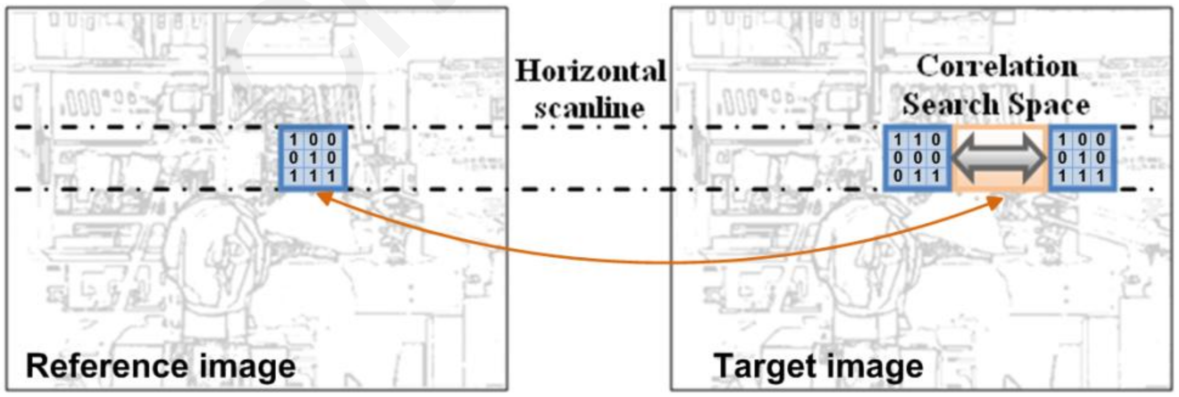
•*THRESSHOLD*: The result of the convolution operations (outputs of the MUL_ADD units) is compared with a predetermined threshold value. Comparison returns 1-bit pixel intensity values, which are concatenated into two 16-bit registers. The values from the 16-bit registers are forwarded to the output port (FIFO queues of the DCU) once every 8 clock cycles.

## 3.4 Disparity Computation Unit (DCU)

The DCU calculates sparse disparity maps covering disparity ranges up to 120 pixels and correlation window sizes from 3x3 to 11x11. Figure 3.1 (b) shows the architecture of the DCU, which involves three major steps: *INPUT/OUTPUT*, *SADs_COMPUTATION* and MIN*SAD_COMPUTATION*. The architecture consists of an I/O controller that reads/writes data from/to the I/O port, on-chip memory (FIFO queues, scanline buffers and window buffers) for temporarily storing the edge data, and a unit that searches and finds the position of the edge to be processed (edge tracking unit - ETU). It also consists of a collection of adders and subtractors to compute the SAD values for all disparity levels, a unit to compute the minimum SAD value, multiplexers, and intermediate pipeline registers. Apart from reading/writing data from/to the I/O port, the I/O controller also acts as a control unit, coordinating memory accesses from the on-chip memory, as well as data transfers and handshakes between the components of the DCU.

### 3.4.1 DCU process overview

The DCU performs correlation on the binary image pair produced by the EDU. Correlation works by comparing the edge points of the two images found in the scanline buffers. The correlation values are computed using the SAD correlation metric, which in the case of binary data, is reduced to a hamming distance operation that can be directly implemented in hardware using only addition and 1-bit subtraction operations. The comparison of the edge points is carried out by a correlation window from the reference image (fetched from the reference image scanline buffers and stored to the window buffer), that is correlated with a second window from the target image (fetched from the target image scanline buffers and stored to the disparity range buffer). The second window is shifted through all possible positions in the target image. These positions are bounded by the disparity range ($d_m$ – $d_M$). The location where the correlation search yields the best score (minimum in our case), determines the edge in the target image that best corresponds to the edge of interest in the



$$\text{SAD: } \Sigma|T_i-R_i| = |1-1|+|0-1|+|0-0|+|0-0|+|1-0|+|0-0|+|1-0|+|1-1|+|1-1|$$
$$= 0+1+0+0+1+0+1+0+0=3$$

**Figure 3.3: Search area and SAD technique used in correlation matching.**

reference image, and consequently yields a potentially matching edge in the two images.

Figure 3.3 shows the correlation search space and the SAD matching technique between correlation windows. Each window in the target image is shifted on a predetermined offset (one pixel in our implementation), and compared to the correlation window in the reference image. The process is repeated for all disparity levels (i.e. disparity range, between 1 and 120 in our case). Something that needs to be further discussed is the locations that the correlation window in the reference and target images is allowed to move, and subsequently the value of the central pixel of the correlation window in the reference and target images. As mentioned earlier, the correlation window in the reference image is moved only to the edges (edge points are represented with 1 and the non-edge points with 0). Therefore, the central pixel of the correlation window in the reference image always has a value of 1. The central pixel of the correlation window in the target image, on the other hand, can take a value of either 0 or 1, since the window is moved in one pixel increments. The proposed system, therefore, combines characteristics from both block- and feature- matching stereo correspondence algorithms, to increase the frame rate.

### 3.4.2 DCU Architecture Overview

The DCU unit was designed with emphasis on parallelism, targeting a large number of search and match operations performed in a single clock cycle. This is facilitated by the simplicity of the adders and subtractors used (due to the use of binary data), and by the organization of the adders and comparators in tree structures. The DCU exploits both pixel-level and window-level parallelism [**120**]. Pixel-level parallelism is achieved by computing the absolute differences in a SAD in parallel. Window-level parallelism is achieved by computing the SAD values for all disparity levels in a pipelined and parallel manner. The number of SAD values that are computed in parallel depends on the targeted operating frequency of the DCU. The same happens for the *MINSAD_COMPUTATION* stage, where the minimum disparity value is computed. As the performance of the DCU depends on the time necessary to compute the edge points, the level of parallelism for the DCU is somewhat restricted by the speed of the EDU and its ability to generate features (edges). However, the EDU can be parallelized as well, so the real limitation in terms of parallelism is the external memory I/O.

Due to its pipelined and parallel structure, the DCU presents good scalability in terms of correlation window size and disparity range. Particularly, it can compute the SAD values for all possible positions of the shifting window with a maximum size of 11x11 in two clock cycles, and the minimum SAD value for a maximum of 120 disparity levels in three cycles. However, once the pipeline fills up and the FIFO queues are not empty, the DCU can provide a disparity value at the output every clock cycle.

To keep a constant flow of data in the pipeline, the DCU must be able to locate one edge in the reference image every clock cycle, while discarding the non-edge points found between successive edges. At the same time, the DCU must have parallel access to the *mxm* window surrounding the edge found in the reference image, as well as to the corresponding $d_M$ windows from the target image. For these reasons, each scanline buffer used in the DCU consists of a series of 16-bit registers, and can store *m* scanlines from an input image. We avoid using 1-bit registers in order to facilitate more parallelism and to make the process of discarding the non-edge points fast. The 16-bit registers are organized into FIFO structures, and allow parallel access to their elements. Specifically, the scanline buffers for the reference image (RSB) output 16 successive *mxm* windows (stored in the Candidate Window buffers), while the scanline buffers for the target image (TSB) output 16 successive $m\text{x}(m+d_M)$ windows (stored in the Candidate Disparity Range buffers). The RSB also output a 16-bit vector (search vector) from positions $1+w$ to $16+w$ of the $(w+1)^{th}$ scanline, where $w=(m-1)/2$. The search vector is being searched for potential edge points by the edge tracking unit (ETU), which is the connection point between the *INPUT/OUTPUT* stage and the remaining stages. The ETU works by locating an edge and its corresponding position in the 16-bit search vector every clock cycle. The positions of the edges found during the searching process are used to select the window and disparity range buffers (among the 16 candidates) corresponding to the edge points found; the selected buffers become the input of the next pipeline stage. It must be noted that the ETU requires from 1 cycle (in the best case) to 16 cycles (in the worst) to locate all edges in the search vector. During this period, the *edge_found* signal is set to 1 and the scanline buffers are disabled, so that the content of the candidate window and candidate disparity range buffers remains constant. When all edges in the search vector are located, the ETU sets the *edge_found* signal to 0. This informs the I/O controller to fetch new edges from the input FIFO queues and to shift the scanline buffers to the right.

A description of each major step involved in the DCU is given next:

•*INPUT/OUTPUT*: This pipeline stage fetches pixel data from the I/O port, executes the edge tracking process, and selects the windows corresponding to the edges found. The data fetched from the input port (the two 16-bit vectors produced by the EDU) are stored into FIFO queues. The I/O controller reads data from the input FIFO queues (if they are not empty) and forwards data to the scanline buffers (16-bits to each scanline buffer), until the first *m* scanlines from both images are stored into the scanline buffers. After the scanline buffers are filled, the I/O controller reads new pixel data from the queues only if the *edge_found* signal is set to 0 by the ETU. While the *edge_found* signal is set to 1, the scanline buffers are disabled, and during this period the edge tracking process described above is performed. If there is new data available at the input during this period, this data is written to the input queues. Furthermore, during this pipeline stage, the I/O controller writes the disparity value computed in the previous cycle to the output port.

•*SADs_COMPUTATION*: The SAD values for all disparity levels are computed during this pipeline stage. The stage consists of $d_M$ Absolute Difference (ABDIF) units, which compute the absolute difference between the *mxm* correlation window (stored in the Window Buffer) and the $d_M$ *mxm* windows (stored in the Disparity Range Buffer). Each ABDIF unit receives as input two $m^2$-bit vectors, whose elements are the edge points of the correlation windows, and consists of $m^2$ 1-bit subtractors that compute the absolute difference of the edge points. The output of each ABDIF unit is an $m^2$-bit vector, which is next added bitwise using binary tree adders (BTA). Given the 11x11 maximum supported correlation window size, and 1-bit pixel intensities, the maximum value of the addition operation cannot be greater than 121. As such, the outputs of the BTA units are 7-bit values.

•*MINSAD_COMPUTATION*: The SAD values for all disparity levels in the range [1 : $d_M$] are compared with each other in order to compute the minimum value and its disparity. The comparison is carried out by a collection of 7-bit comparators and registers, arranged in tree structure to reduce the delay of the longest path. As stated previously, this stage was further divided into 3 pipeline stages in order to meet the targeted operating frequency (100 MHz). Figure 3.4 shows the circuit that computes the minimum SAD value and its disparity. Each minSAD unit receives as input two 14-bit vectors, each of which is a concatenation of a SAD value (7 bits) and its corresponding disparity (7 bits - up to 120 disparity levels). The minSAD unit compares the two SAD values and outputs the minimum of them along with its disparity. The entire circuit for computing the minimum SAD value and its disparity consists of multiple minSAD units, arranged in a structure of a binary tree of $\log_2(d_M)$ levels. The circuit is configurable in terms of the disparity range, allowing the upper bound ($d_M$) of the disparity range to lie in the range [1:120]. In the probable cases, however, where $d_M$ is less

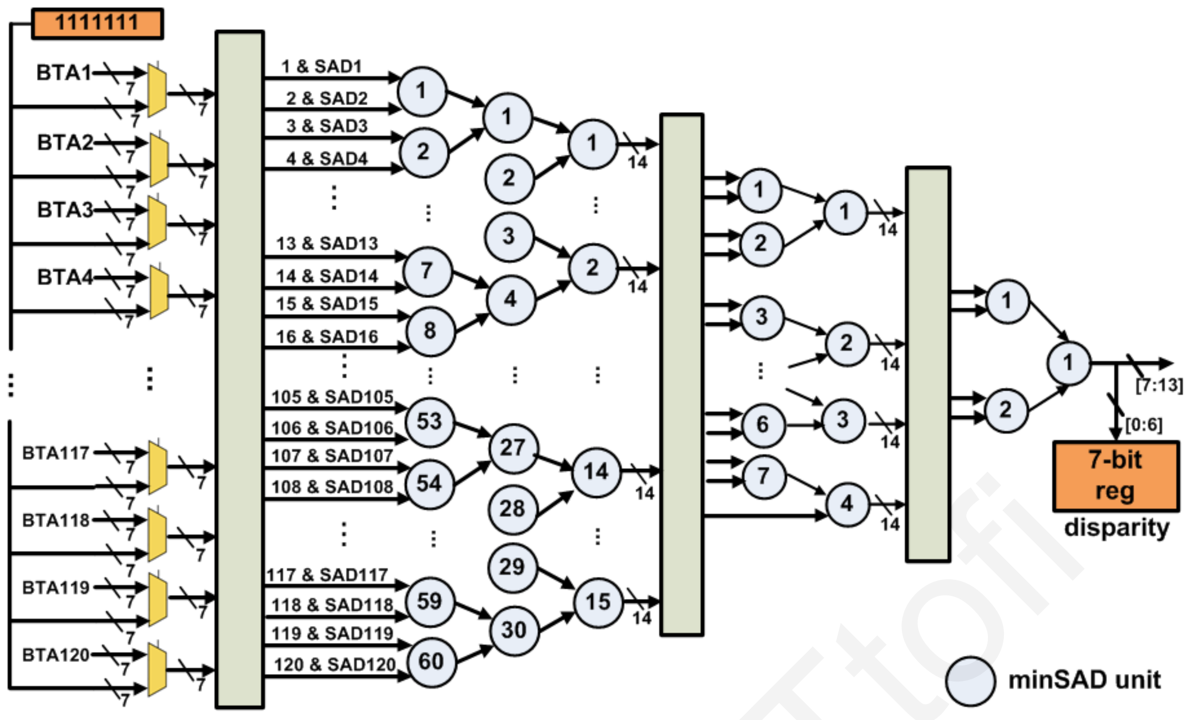


Figure 3.4: Computing the minimum SAD value.

than 120, values between $d_M$+1 and 120 are not valid disparity values, and the circuit must have a way to ignore them and maintain temporal consistency. This is achieved by the 2-to-1 multiplexers located at the outputs of the BTA units. Each multiplexer receives as inputs the output of a BTA and the value of 127 (maximum value for a 7-bit number). The control logic for the multiplexers is generated by the I/O controller (control unit) using a decoder to decode the $d_M$ value into 120 control signals. In this way, the disparity values between $d_M$+1 and 120 are not taken into account during the process of finding the minimum SAD value, while maintaining correct operation.

## 3.5 Experimental Platform and Results

### 3.5.1 Experimental Platform

We developed a prototype of the architecture shown in Figure 3.1 using the Xilinx ML505 Evaluation Platform [**121**], which is equipped with a Virtex-5 LX110T FPGA and features several I/O ports suitable for experimental evaluation. We used the Microblaze soft-processor (provided with Xilinx EDK tool) as the Memory Controller & Control Unit (MCCU) of the system [**122**]. Microblaze is used to handle tasks such as system I/O and control, stereo image rectification, as well as data transfers and handshakes from/to the EDU and the DCU. It is also used as a parameter initialization mechanism, initializing the system architecture parameters, such as correlation window size and disparity range. It communicates with external devices through the Processor Local Bus (PLB) for data transfers and monitoring purposes. It also communicates with the EDU and DCU using a FIFO-based communication protocol over its accelerator interface, called Fast Simplex Link (FSL). The data transfers over the FSL are non-blocking, allowing thus the communication to be overlapped with the computation performed at the EDU and the DCU. Additionally, the Microblaze implements the interpolation mechanism that is used to fill each non-edge point of the sparse disparity map generated by the DCU. Figure 3.6 (a) shows a block diagram of the particular system implemented on the FPGA platform.

The system was evaluated using both synthetic and real-world data. The synthetic data includes 10 stereo images from the Middlebury database [**25**], [**26**], while the real-world data includes stereo images taken in the laboratory and from a moving vehicle using a night vision stereo camera (provided in [**27**]). We used the Microblaze processor to generate the rectified real-world images, which were stored in the on-board DRAM, and fed as input to the system shown in Figure 3.1. The system was visually verified by displaying the resulting disparity

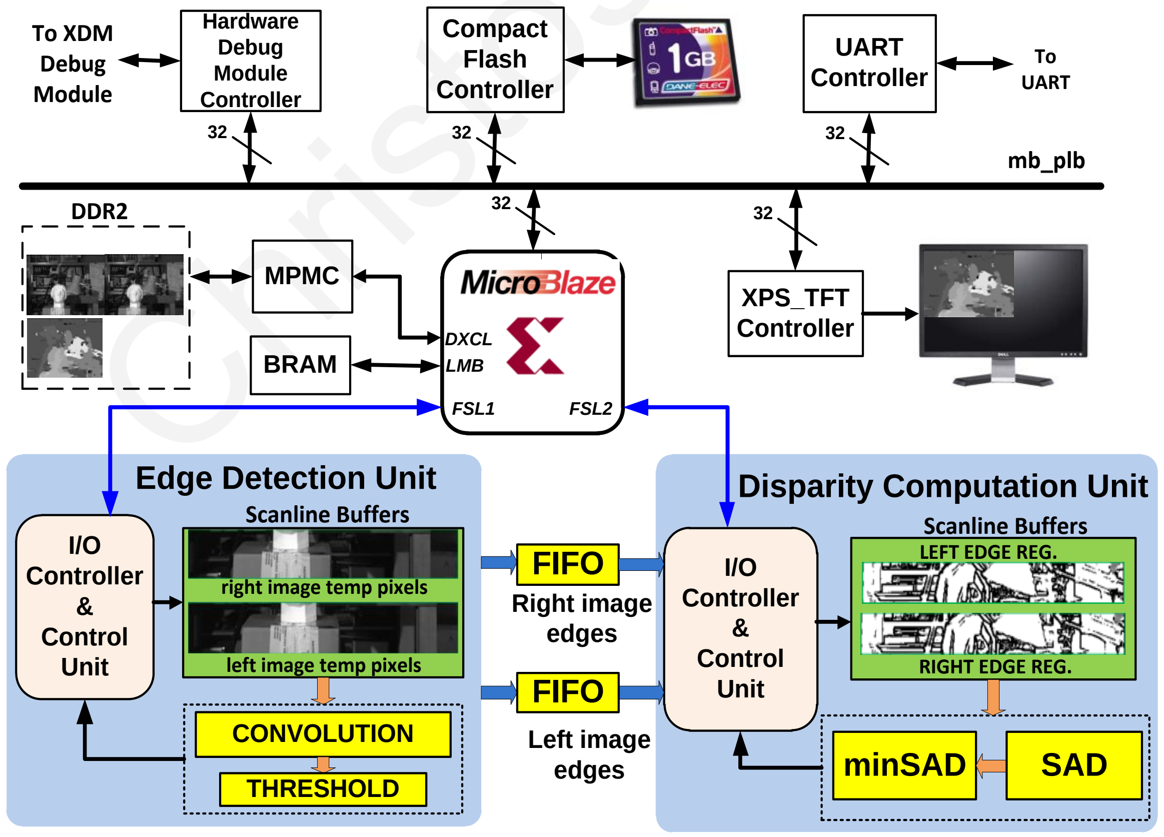


**Figure 3.5: FPGA implementation used to verify the proposed architecture.**

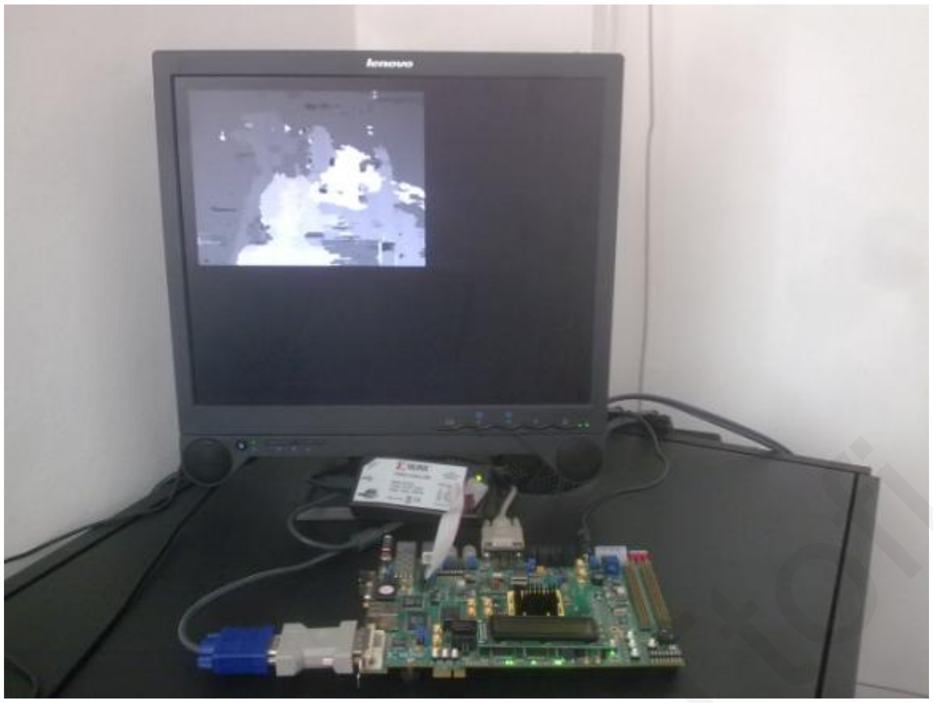
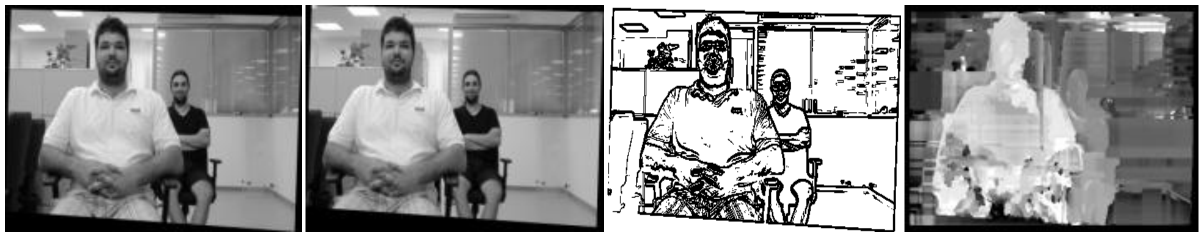


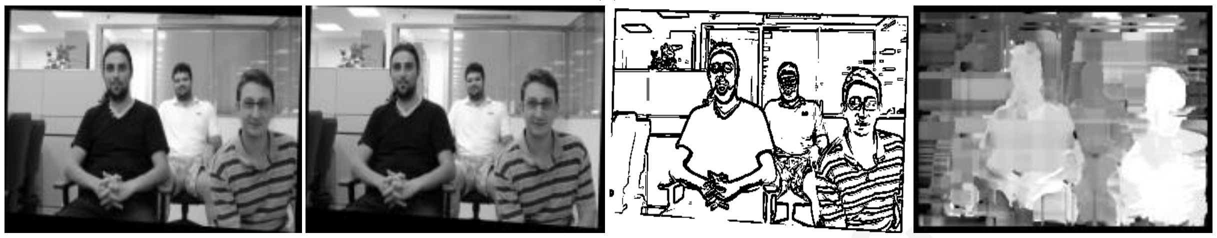
**Figure 3.6: FPGA implementation used to verify the proposed architecture.**

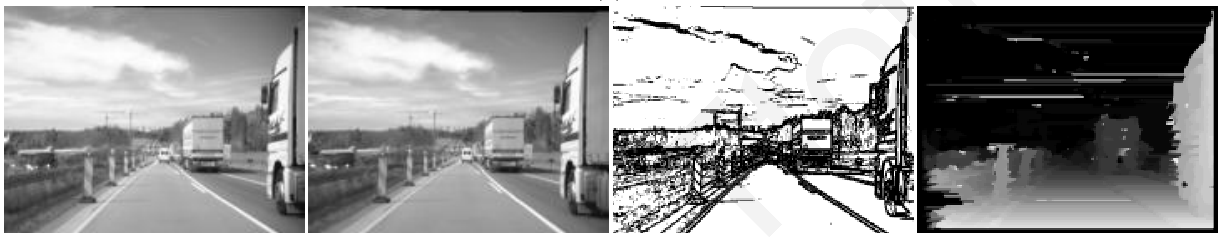
maps to a thin-film transistor (TFT) monitor. Figure 3.6 (b) illustrates the experimental setup of the system and an example display. The evaluation results for the synthetic images are shown in Table 3.1 and Table 3.2 for the Tsukuba, Venus, Teddy and Cones image pairs (for which there is an online evaluation system [**25**], [**26**]). Figure 3.7 shows the rectified image pairs generated by the Microblaze processor, the output of the Sobel detector (for the left images), and the resulting disparity maps for the real-world stereo images.

### 3.5.2 Disparity Map Quality–Impact of Edge Detector

To evaluate the quality of the proposed system and examine the impact of the edge detection algorithm in the overall system quality, we use stereo pairs from the Middlebury database [**25**], [**26**] for which the ground truth disparity maps are known, and measure the incorrect disparity estimates using the percentage of bad pixels evaluation metric [**26**]. We measure the percentage of bad pixels for both the sparse disparity maps generated by the DCU, as well as the dense disparity maps obtained using nearest neighbor interpolation. When evaluating the sparse disparity maps, we account only for the disparity estimates at edge points, whereas, when evaluating the dense disparity maps, we account for the disparity estimates at all image points.


(a)


(b)


(c)

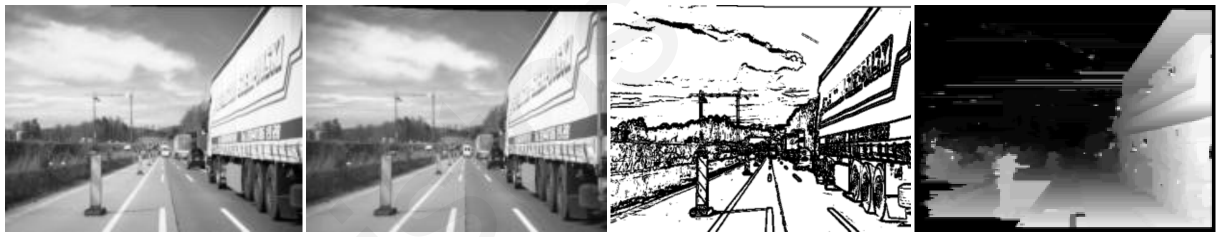

(d)

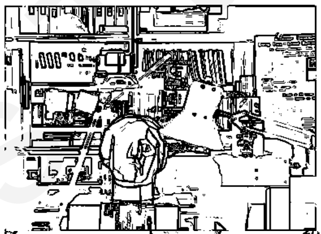
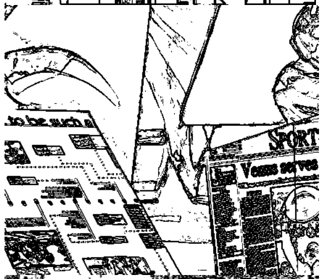
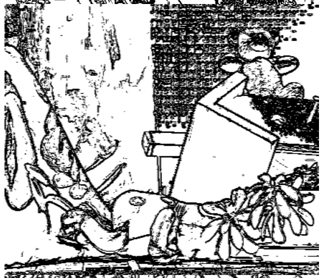
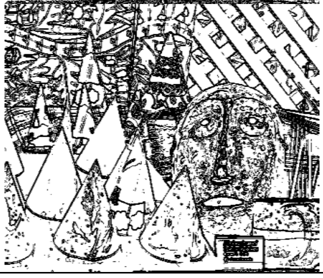
**Figure 3.7: Evaluation results for real-world images.**

To describe the percentage of bad pixels for both cases, we adopt a notation where the subscript, called *P*, identifies the set of points for which a disparity value is estimated i.e. $P \in \{E, A\}$ (*E*=edge points, *A*=all image points). The percentage of bad pixels, $B_P$, ($B_E$ and $B_A$ refer to the percentage of bad pixels for the set of edge points and for all image points, respectively), is then given by (3.1), where $d_C(x,y)$ and $d_T(x,y)$ are the computed and ground truth disparity maps, respectively. $N_P$ is the total number of pixels in the set of points *E* or *A* respectively, and $\delta_d$ is the disparity error tolerance. We set $\delta_d$ = 1.0 for all experiments we carried out as this value coincides with some previously published works [**105**], [**92**], [**107**], [**108**], [**114**], [**109**].

$$B_P = \frac{1}{N_P} \sum_{(x,y) \in P} (|d_c(x,y) - d_T(x,y)| > \delta_d), \quad P \epsilon \{E, A\} \tag{3.1}$$

We first measured the percentage of bad pixels incurred from the use of the three different edge detectors (Canny, Sobel, evolvable), in order to examine the efficiency of each detector in terms of accuracy and speedup. This was achieved by varying the threshold of each detector until finding the one that yields the minimum percentage of bad pixels. The data reduction associated with the minimum percentage of bad pixels for each detector was also recorded, as the speedup improvement of our method is strongly related to the data reduction. Experimental results were carried out for ten image pairs from the Middlebury dataset [**25**], [**26**], indicating an average percentage of bad pixels (over all image points) of 23.3%, 20.3%, and 18.9% for the Canny, Sobel and evolvable edge detector, respectively. Table 3.3 shows the percentage of bad pixels for a sample of 4 representative pairs. As it can be seen in Table 3.3,

**Table 3.1: Input image pairs and the output of the EDU for the left image.**

| Left image | Right image | Edge image (left image) | % data reduction |
|---|---|---|---|
| | |  | 70.23% |
| | |  | 73.70% |
| | |  | 67.33% |
| | |  | 65.85% |
| **Average percentage of data reduction = 69.3%** | | | |

the detector that yields the better quality is the evolvable. However, this detector obtains the smallest amount of data reduction among all detectors. The Canny detector, on the other hand, although yields the larger data reduction, its accuracy is less than the other detectors. For these reasons, we chose the Sobel detector.

Experimental results carried out over the ten image pairs indicate that the Sobel edge detector can reduce search data on an average of 55-85%, depending on the content of the images and the threshold used by the detector. The impact of the threshold value was determined by measuring BA for a wide range of threshold values. Figure 3.8 illustrates how the percentage of bad pixels changes with respect to the threshold value for four sample images from the Middlebury database (Tsukuba, Venus, Teddy, Cones). The results indicate

**Table 3.2: Evaluation results of the proposed system using Middlebury stereo pairs.**

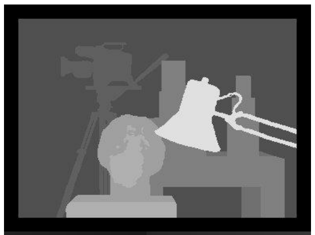
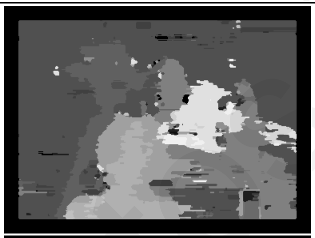
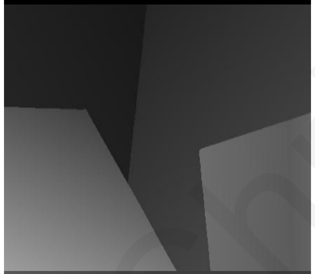
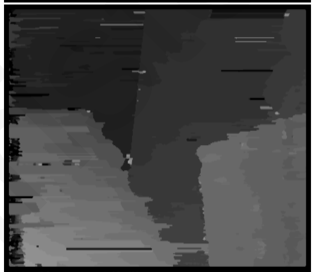
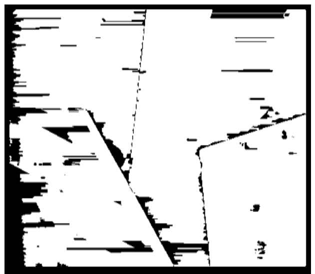
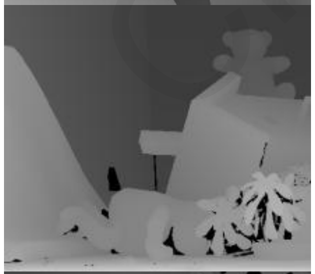
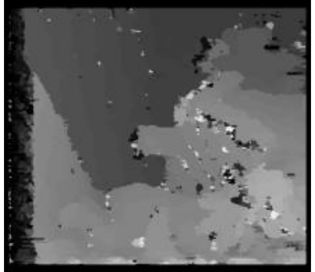
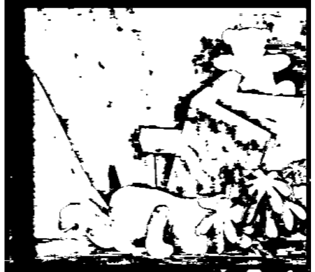
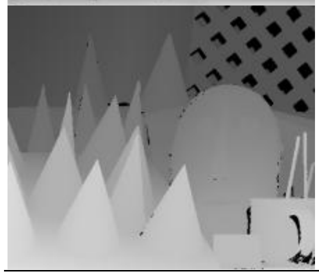
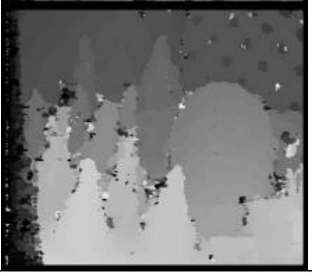
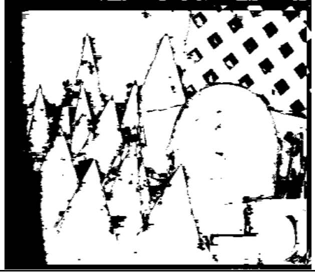
| Ground truth | Computed disparity map | Bad pixels ($\delta_d$=1, $B_A$) | % of bad pixels |
|---|---|---|---|
|  |  | | nonocc = 9.26% all = 10.4% disc = 28.2% |
|  |  |  | nonocc = 11.0% all = 12.1% disc = 28.9% |
|  |  |  | nonocc = 21.4% all = 29.1% disc = 41.3% |
|  |  |  | nonocc = 17% all = 25.3% disc = 33.4% |
| | | **Average percentage of bad pixels = 22.3%** | |

**Table 3.3: Quality reduction for different edge detectors.**

| Image pair | Detector | | | | | |
|---|---|---|---|---|---|---|
| | Canny | | Sobel | | Evolvable | |
| | *$B_A$* | *% of data reduction* | *$B_A$* | *% of data reduction* | *$B_A$* | *% of data reduction* |
| **Tsukuba** | 10.30 | 82.95 | 8.73 | 70.23 | 11.04 | 57.67 |
| **Venus** | 17.03 | 81.08 | 16.73 | 73.70 | 12.81 | 55.78 |
| **Teddy** | 31.57 | 81.35 | 30.38 | 67.33 | 26.78 | 50.33 |
| **Cones** | 29.65 | 80.64 | 27.60 | 65.85 | 24.47 | 52.75 |

that the optimal threshold value lies in the range 0.005-0.015. We found experemintally that threshold values in this range work well for the real-world images as well.

We next measured the percentage of bad pixels imposed by the use of the Sobel edge detector (using the optimal threshold values). Simulation results indicate that the average percentage of bad pixels is ~20.33% when considering all image points (dense maps were obtained with interpolation), and ~7.2% when considering only the edge points (sparse disparity maps). Table 3.1 shows four sample input image pairs used for evaluation, along with the output of the Sobel edge detector (using the optimal threshold value for each pair). Correspondingly, the evaluation results using a correlation window size of 11x11 and the optimal detector threshold value for each pair are shown in Table 3.2. The table shows the ground truth disparity map for each sample input image pair (column 1), the dense disparity maps generated by the proposed system after interpolation (column 2), and the percentage of bad pixels, $B_A$, when $\delta_d$=1.0 (column 3). The table also lists the percentage of bad pixels for three different kinds of regions; the non-occluded regions (nonocc), the half-occluded regions (all), the depth discontinuity regions (disc), as well as the average percentage of bad pixels (column 4). All statistics in Table 3.2 (column 4) were computed using the Middlebury online evaluation system [**25**], [**26**]. The percentage of bad pixels, $B_E$, (for the set of edge points only) for the same input image pairs are shown in Table 3.4. An important observation regarding $B_A$ and $B_E$, is that the disparity estimates at edge points are more accurate (present smaller percentage of bad pixels), as obviously anticipated.

**Table 3.4: Average percentage of bad pixels $B_E$.**

| Image pair | Tsukuba | Venus | Teddy | Cones |
|---|---|---|---|---|
| **$B_E$** | 2.31% | 3.11% | 14.03% | 12.55% |
| **Coverage** | 29.77% | 26.30% | 32.67% | 34.15% |
| **Accuracy (100 - $B_E$)** | 97.69% | 96.89% | 85.97% | 87.45% |

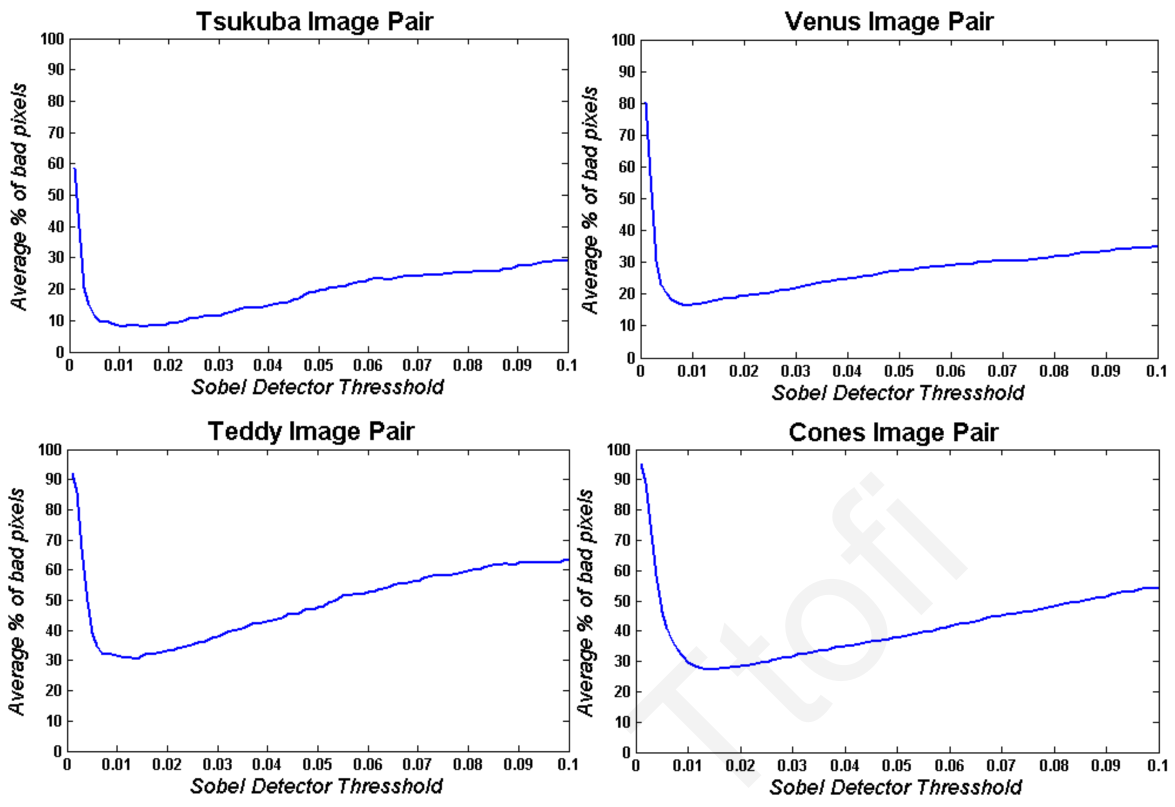


**Figure 3.8: Percentage of bad pixels vs. Sobel threshold.**

### 3.5.3 Disparity Map Quality Analysis

For a detailed quality analysis, we compare the quality of the disparity maps generated by the proposed method to the disparity maps generated by a stand-alone SAD block matching method. The latter consists only of a DCU having a similar architecture with the DCU shown in Figure 3.1 (b) (without the ETU and the candidate window and disparity range buffers), which is able to process 8-bit pixel values instead of binary data (edges). The results, extracted by comparing the disparity maps of both methods to the ground truth disparity maps, are presented in Table 3.5 (last two rows) for the four sample input images. The dense disparity maps generated by the proposed method (using nearest neighbor interpolation) are more accurate, as the average percentage of bad pixels is 22.3%, whereas the percentage of bad pixels of the stand-alone SAD block matching method is 25.3%. This is because stand-alone SAD suffers from changes in pixel intensities and is sensitive to depth discontinuities and to regions of uniform textures in images as well [**60**]. On the other hand, the proposed hybrid method potentially tackles these problems, by limiting the correspondence search to specific reliable features in the images (edges in our case).

Furthermore, we compare the quality of our method with some related works. The works

Table 3.5: Quality comparison of the proposed method with other methods.

| Image Pair | Image Region | Georgoulas [23] | Yang [33] | Darabiha [36] | Jin [37] | Chang [38] | Ambrosch [42] | Stand-alone SAD | Edge-Directed |
|---|---|---|---|---|---|---|---|---|---|
| Tsukuba | nonocc | N/A | 1.49 | 19.59 | 9.79 | N/A | 5.81 | 8.40 | 9.26 |
| | all | 13.55 | 3.40 | N/A | 11.56 | 2.80 | 7.14 | 10.1 | 10.4 |
| | disc | N/A | N/A | 37.62 | 20.29 | 20.29 | 22.6 | 34.3 | 28.2 |
| Venus | nonocc | N/A | 0.77 | 10.51 | 3.59 | N/A | 2.61 | 6.27 | 11.0 |
| | all | 12.6 | 1.90 | N/A | 5.27 | 0.64 | 3.33 | 7.86 | 12.1 |
| | disc | N/A | N/A | 31.52 | 36.82 | N/A | 25.3 | 47.8 | 28.9 |
| Teddy | nonocc | N/A | 8.72 | N/A | 12.50 | N/A | 9.79 | 25.6 | 21.4 |
| | all | N/A | 13.2 | N/A | 21.50 | 13.7 | 15.5 | 33.2 | 29.1 |
| | disc | N/A | N/A | N/A | 30.57 | N/A | 25.7 | 45.4 | 41.3 |
| Cones | nonocc | N/A | 4.61 | N/A | 7.34 | N/A | 5.08 | 18.7 | 17 |
| | all | 12.6 | 11.6 | N/A | 17.58 | 10.1 | 11.5 | 27.7 | 25.3 |
| | disc | N/A | N/A | N/A | 21.01 | N/A | 15.0 | 38.2 | 33.4 |
| Average % of bad pixels | | N/A | 7.69 | N/A | 17.24 | N/A | 12.5 | 25.3 | 22.3 |

in [**91**], [**123**], [**124**], [**125**] and [**126**] are omitted, either because they do not present quality results, or because the quality metric used is different from the one adopted in this work. The results are presented in Table 3.5. As it can be seen, the percentage of bad pixels for our method is comparable to the majority of local methods ( [**105**], [**107**], [**108**]), but it presents a reduction compared to local methods such as [**114**] and [**109**]. Both [**114**] and [**109**], however, achieve lower frame rates; they achieve real-time performance for small-sized (352x288) and intermediate-sized (750x400) images, respectively. Furthermore, [**114**] focuses only on synthetic image data. As evidenced in [**29**], however, methods that work well on synthetic scenes might not work well on real-world scenes. Lastly, our method presents an expected quality reduction compared to global methods such as hierarchical belief propagation [**92**]. Despite the quality reduction, the disparity maps generated by the proposed system still preserve most of the details of the disparity map values. As shown from the performance results (Table 3.6 & Table 3.7 in Section 3.5.4), the reduction in quality can be offset by the high output frame rates of the system, depending on the application demands. Some applications with hard real-time constraints (e.g. obstacle and object detection) may work with lesser-quality disparity maps [**45**], but benefit from the high-performance in order to perform the required tasks on-time. Interested readers are referred to Chapter 6.1 (or reference [**127**]) for a successful application of the proposed method to a hardware object detection system.

The quality of the proposed system is also affected by the chosen interpolation mechanism, which generates dense disparity maps (see Section 4.1). As stated before, we implement the nearest-neighbor mechanism; even though a bicubic interpolation mechanism can perform better with certain images, the nearest neighbor mechanism performs significantly better with the real-world images used in this work. Images (a) and (b) shown in Figure 3.7 were computed using bicubic interpolation, whereas images (c) and (d) were computed using nearest-neighbor. These results indicate that further experimentation is necessary to determine the interpolation mechanism, which however, is also impacted by the environment that the host application will operate. We plan to experiment with other interpolation mechanisms in future work, and evaluate them both in terms of quality but also in terms of their performance and hardware requirements.

### 3.5.4 System Performance

The performance, in frames per second (fps), of the proposed system architecture mainly depends on the delay overheads incurred from the Sobel edge detection and the disparity computation units. There are many factors affecting the resulting system frame rate, including the data reduction obtained from the edge detector (*the percentage of non-edge points over the total image points*), the image size and the I/O bandwidth from/to the external memory. Obviously, there is a performance improvement if the time saved by the stereo correspondence algorithm due to data reduction is more than the delay overheads incurred from the edge detector. With respect to the image size and the I/O bandwidth, the performance decreases as the image size increases and the I/O bandwidth decreases, since in the former case there is more data to be processed, while in the latter case, the amount of data flowing into the system limits the system throughput.

To evaluate the performance of the proposed architecture, we setup two different system configurations: one based on the proposed edge-based method, and one using a stand-alone SAD-based approach. The two systems have the same I/O constraints, memory requirements and operating frequency, and support the same correlation window sizes and disparity levels ranges. We identify the speedup of the proposed architecture compared to the stand-alone SAD approach, and provide results when increasing the input image size in order to illustrate the

**Table 3.6: Image size and system performance.**

| Images size | Max. Performance (fps) | | | | Performance (fps) of the FPGA Prototype |
|---|---|---|---|---|---|
| | EDU | DCU | Stand-alone SAD | Overall System | |
| **320x240** | 2572 | 3185 | 1225.5 | 2570 | 857 |
| **640x480** | 647 | 1291 | 315 | 645.4 | 215 |
| **800x600** | 414.6 | 730.8 | 203.3 | 412 | 138 |
| **1024x768** | 253.3 | 536.2 | 124.7 | 252 | 84 |
| **1280x1024** | 152.1 | 337 | 75.2 | 150.4 | 50 |

* OPERATING FREQUENCY = 100 MHz

Table 3.7: Comparison of MDE/s performance for various systems and methods.

| Work | Image size | Disparity Range | Frame rate (fps) | MDE/s | Algorithm | Platform |
|---|---|---|---|---|---|---|
| **Q. Yang [33]** | 384x288 | 16 | 12.77 | 22.2 | Hierarchical belief propagation | NVIDIA Geforce 7900 GTX GPU |
| **Diaz [35]** | 1280x960 | 29 | 52 | 1885 | Phase based | Custom FPGA, Xilinx Virtex-II (65MHz) |
| **Darabiha [36]** | 256x360 | 20 | 30.3 | 55.2 | LWPC (phase correlation) | TM-3A board (Xilinx Virtex-4 2000E FPGA) |
| **Jin [37]** | 640x480 | 64 | 230 | 4522 | Census transform | Virtex-5 XC4VLX200-10 FPGA (93.1MHz) |
| **Ambrosch [31]** | 450x375 | 100 | 600 | 10125 | Block-matching (SAD) | EP2S130 (110MHz) |
| **R. Yang [7]** | 512x512 | 32 | N/A | 289 | Block-matching (SAD) | ATI Radeon 9800 graphics card |
| **Hile [8]** | 512x480 | 32 | 30 | 235.9 | Block-matching (SAD) | N/A |
| **Miyajima [9]** | 640x480 | 80 | 26 | 639 | Block-matching (SAD) | ADM-XRC-II (40MHz) |
| ***Arias-Estrada* [11]** | 320x240 | 16 | 71 | 87.2 | Block-matching (SAD) | XCV800HQ240-6 (66MHz) |
| **Lee [12]** | 640x480 | 64 | 30 | 589 | Block-matching (SAD) | XC2V8000 (10MHz) |
| **Hariyama [13]** | 64x64 | 64 | 5063 | 1327.2 | Block-matching (SAD) | APEX20KE (86MHz) |
| **Georgoulas [23]** | 800x600 | 80 | 550 | 21120 | Block-matching (SAD) | EP4SGX290 (511MHz) |
| **N. Y. Chang [38]** | 352x288 | 64 | 42 | 272.5 | Mini-census adsw | UMC 90ns Std. Cell |
| **K. Ambrosch [42]** | 750x400 | 60 | 60 | 1080 | SAD-IGMCT | Altera Stratix I (133MHz) |
| **Proposed** | 1280x1024 | 120 | 50 | 7864 | Hybrid method ( block- and feature-matching) | Virtex-5 LX110T FPGA (100 MHz) |

impact of the edge detector as the image size increases.

We used the 10 synthetic stereo image pairs as benchmarks in order to extract performance results for both systems mentioned above. The results are shown in Table 3.6. Columns 2-5 of the table provide the maximum performance achieved when the I/O bandwidth is 32-bits/cycle. These columns illustrate the impact of the input image size on the performance of the EDU and DCU, as well as on the average system performance for both system configurations (proposed system vs. stand-alone SAD). As it can be seen, the performance is inversely proportional with the input image size in both cases. The use of the edge detector, however, almost doubles the average system performance of the proposed method, when compared to the stand-alone SAD method, for all images sizes. This is due to the search data reduction. Table 3.6 also illustrates how the performance of the proposed system is affected by the data reduction and the frame rates of the EDU and DCU units. While the speedup obtained by the DCU lies between 2.2 and 6.5 (due to the 55%-85% data reduction), the proposed method obtains an overall speedup of ~2, because of the performance of the EDU (in this case, it is the bottleneck). This of course can be improved by increasing the parallelism inside the EDU through higher I/O bandwidth. It must be noted that, in the case where the percentage of data reduction is much smaller than 55% (the input images contain many more features), then the bottleneck of the system will become the DCU. The data reduction, however, is rarely less than 55%, as in the average natural scene, edges mostly signify object boundaries, which cover only a small part of an image as opposed to the “body” of objects [**118**].

The last column of Table 3.6 gives the performance of the prototype system implemented on the FPGA platform. The performance was computed through a dedicated hardware counter that measured the number of cycles for all images; we obtained the average frame rate for all 10 stereo pairs. The performance of the prototype system is lower than the maximum performance (shown in column 5), as it is mainly limited by the ability of the Microblaze processor to fetch data from the external DRAM. Even with the limitation of the Microblaze, the prototype implementation still achieves real-time frame rates even for high-resolution images. We plan to implement the external memory controller as a custom hardware unit in the future, so that to avoid the limitation of using an embedded processor, and benefit from the higher frame rates achieved when the I/O bandwidth is 32 bits/cycle.

Table 3.7 presents a comparison between existing implementations [**91**], [**123**], [**124**], [**125**], [**128**], [**120**], [**105**], [**106**], [**92**], [**126**], [**107**], [**108**], [**114**], [**109**] and the proposed FPGA prototype. Performance is provided in frames per second (fps), as well as in Million Disparity Estimations per second (MDE/s). The proposed system is faster than all implementations when considering the input image size and the maximum disparity range supported, primarily due to the reduction of the search space obtained by the integration of the edge detector. Moreover, the proposed system achieves a performance of 7864 MDE/s, which is among the highest rates reported. The only works that achieve higher MDE/s is [**105**] and [**106**]. Both of these works, however, provide only synthesis results; the papers do not clarify details about the implementation on the FPGAs and the I/O rate. As we discussed in the previous paragraph, the proposed system architecture can achieve even higher rates by implementing a custom external memory controller (23593 MDE/s). Conclusively, performance results indicate that the proposed system architecture exhibits high potential for applications with hard real-time constraints; the proposed system can meet the real-time requirements of such applications, even for high-resolution images.

### 3.5.5 Hardware Overheads

The proposed FPGA system was evaluated for relevant metrics such as area and operating frequency. Moreover, estimated power consumption (dynamic and static) figures were obtained using Xilinx's XPower analyzer tool, assuming a switching activity of 50%. The impact of the correlation window size, disparity range and image size on the aforementioned metrics is also explored.

Prior to presenting the obtained hardware results, however, we first discuss the mapping on the FPGA fabric, as it highly relates to the resulting performance and area/power costs. The DCU is more demanding than the EDU in terms of the computations performed, but the EDU requires a larger amount of on-chip memory for implementing the scanline buffers for the two input images, as it processes 8-bit values (pixels). The scanline buffers can be mapped on the Virtex-5 FPGA either by utilizing the 1-bit registers included in each FPGA slice (four 1-bit registers per slice), or by configuring the 6-input LUT of each slice as a 32-bit shift register [**121**]. The former consumes considerable amount of area in terms of slice registers; we followed the latter approach in an attempt to keep the area utilization of the EDU minimal. The scanline buffers of the DCU are much smaller, and thus, were mapped on the FPGA slice registers. The remaining FPGA slice registers were utilized for the mask values (in the case of the EDU), the window and disparity range buffers (in the case of the DCU), and for the intermediate pipeline registers. The remaining system components (BTA units, minSAD units, MULT_ADD units, etc) involve simple operations such as additions and comparisons, and were mapped on the LUTs.

Table 3.8 shows synthesis and power results for the DCU for different disparity levels up to 120. There is a linear increase in the utilization of the FPGA LUTs as the disparity range increases. This is attributed to the additional logic in adders/subtractors required, since there are more SAD values to be computed as the disparity range becomes larger. Moreover, more comparator units are needed to compute the minimum disparity value. Apart from the increase in the number of LUTs, there is also a linear increase in the number of slice registers allocated for the disparity range buffers and the intermediate pipeline registers. Table 3.8 also shows the impact on the operating frequency of the DCU as the disparity range increases. The frequency decreases as the disparity range increases, suggesting that if we want to allow more than 120 disparity levels, we need to introduce more pipeline stages at the expense of further hardware overhead.

**Table 3.8: Resource utilization and maximum disparity range.**

***(Window size=11x11, Image size=1280x1024)***

| Disparity Range | 20 | 40 | 60 | 80 | 100 | 120 |
|---|---|---|---|---|---|---|
| Slice Registers (out of 69120) | 28963 41.9% | 29536 42.7% | 30134 43.6% | 30237 43.7% | 30592 44.3% | 31863 46.1% |
| Slice LUTs (out of 69120) | 9211 13.3% | 16467 23.8% | 23357 33.8% | 31504 45.6% | 39255 56.8% | 47331 68.5% |
| Frequency (MHz) | 118.7 | 115.5 | 115 | 109.5 | 109.5 | 109 |
| Dynamic Power (W) | 0.6 | 1.2 | 1.7 | 2.5 | 2.8 | 3.3 |

**Table 3.9: Resource utilization and maximum window size.**

***(Max disparity=120, Image size=1280x1024)***

| Window Size | 3x3 | 5x5 | 7x7 | 9x9 | 11x11 |
|---|---|---|---|---|---|
| Slice Registers (out of 69120) | 9429 13.6% | 14965 21.7% | 20648 29.9% | 26096 37.8% | 31863 46.1% |
| Slice LUTs (out of 69120) | 10701 15.5% | 15033 21.7% | 25615 37.1% | 35519 51.4% | 47331 68.5% |
| Frequency (MHz) | 161.4 | 132.9 | 126.1 | 119.6 | 109 |
| Dynamic Power (W) | 0.6 | 1.0 | 1.7 | 2.5 | 3.3 |

**Table 3.10: Resource utilization and maximum image size.**

***(Max disparity=120, Window size=11x11)***

| Image Width | 320 | 640 | 800 | 1024 | 1280 |
|---|---|---|---|---|---|
| Slice Registers (69120) | 10966 15.9% | 17621 25.5% | 20868 30.2% | 26231 37.9% | 31863 46.1% |
| Slice LUTs (69120) | 47331 68.5% | 47331 68.5% | 47331 68.5% | 47331 68.5% | 47331 68.5% |
| Frequency (MHz) | 109 | 109 | 109 | 109 | 109 |
| Dynamic Power (W) | 3.1 | 3.2 | 3.2 | 3.3 | 3.3 |

Table 3.9 shows synthesis and power results for different correlation window sizes (3x3 to 11x11), while the disparity range and image size are kept constant. The results indicate a significant increase in the utilization of FPGA slice registers as the correlation window size increases. This is because more image lines need to be stored in the on-chip memory (scanline buffers) of the DCU. The size of the correlation window buffers and the intermediate pipeline registers also increases, utilizing more slice registers of the FPGA device. Table 3.9 indicates that the number of LUTs increases as well. Although the number of comparator units remains the same (constant disparity range), computing the SAD values between larger correlation windows requires more adders and subtractors mapped on the LUTs. Additionally, the system clock frequency decreases as the correlation window size increases.

Table 3.10 illustrates the impact of the input image width on the DCU resource utilization. Specifically, image width is a major factor affecting mainly the amount of slice

**Table 3.11: Complete system hardware overheads.**

***(Image size=1280x1024, Max disparity range=120, Window size=11x11)***

| Design Unit | Slice Registers (69120) | Slice LUTs (69120) | Block RAMs (148) | DSP48Es (64) | Freq. (MHz) |
|---|---|---|---|---|---|
| **Microblaze System** | 8562 ~12% | 7754 ~11% | 30 ~20% | 6 ~9% | 161.2 |
| **EDU** | 1008 ~3% | 2812 ~4% | 0 | 0 | 134.6 |
| **ABDIF & BTA** | 0 | 129 | 0 | 0 | N/A |
| **Tree Comparators** | 329 | 1760 | 0 | 0 | 174.3 |
| **DCU** | 31863 ~46% | 47331 ~68% | 0 | 0 | 109 |
| **Entire System** | **41559 ~60%** | **66882 ~83.8%** | **30 ~20%** | **6 ~9%** | **100** |

registers required, since more edge points per scanline need to be stored to the on-chip memory (scanline buffers) of the DCU. However, the amount of LUTs and the operating frequency remain unaffected.

Table 3.11 gives the overall hardware demands associated with each of the implemented components, as well as with the entire system implemented on the FPGA. The entire system utilizes ~83.8% of the FPGA LUTs and ~60% of the FPGA slice registers. With respect to the power consumption, the entire system implemented on the FPGA platform dissipates 5.18 W. The total power consumption consists of dynamic power and static power. The dynamic power, which is the power consumed through the operation of the FPGA device, is approximately 4.094 W. The static power, which is the power consumed regardless of design activity, is approximately 1.086 W. It must be noted, that the most energy-demanding unit is the DCU, as it contributes nearly 63% of the total power consumption, while the EDU and the Microblaze system contribute 4% and 12.5% of the total power consumption, respectively. The static power contributes ~20.5% of the total power consumption.

The last rows of Table 3.8, Table 3.9 and Table 3.10 give the dynamic power consumed by the DCU with respect to various system parameters. The tables indicate that the dynamic power consumed by the DCU increases with an increase in the disparity range, window size or image width (as expected). Even though in some of these cases the operating frequency decreases, the increase in the dynamic power consumption is attributed to the increase in the load capacitance, with respect to the amount of logic in the design. It must be noted that the related works included in Table 7 do not present power consumption results; therefore, comparisons are impractical.

Lastly, we compare the hardware overheads of the proposed architecture to the stand-alone SAD (Section 3.5.4), for a maximum disparity range of 80 pixels, a correlation window of 5x5 and an image size of 640x480. We use these values, as larger sizes for each parameter would result in a stand-alone SAD that does not fit on the specific FPGA due to its higher hardware demands. The proposed technique outperforms the stand-alone SAD in terms of both power consumption and efficient resource utilization; it consumes ~33% less power and utilizes ~78.6% and 58.6% slice registers and slice LUTs, respectively. Additionally, we estimate the total energy consumption per frame for each system; the proposed system requires ~69% less energy to process a frame (stereo pair), as it consumes less power in less time (almost half) to process a stereo pair.

## 3.6 Concluding Remarks

Finding corresponding points in stereo images is a tedious task that involves a significant amount of search and match operations. Therefore, it is a major challenge to extract depth information in real time and under limited power / hardware resources. This Chapter presented a stereo vision architecture that relies on the utilization of edge information as a means to reduce the search space involved in stereo matching considerably, thus enabling higher performance rates. The developed approach is basically a hybrid method for solving the stereo matching problem. Through the edge detector that generates the features (edges in our case), over which SAD-based correlation is then applied. The edge detector generates a black and white image pair that characterizes only the object boundaries s of the input image pair. Hence, the stereo matching process does not need to exhaust the entire search space in order to find matching points between the two images. Instead, the correlation window in the reference image is moved only to the edges, providing an excellent way reduce the search space and speed up the overall operation, while also reducing the datapath and on-chip memory requirements. The Chapter discussed hardware-specific issues that must be tackled in order to take advantage of the edge-directed approach and associated benefits, and presented the necessary hardware design optimizations (memory structures needed, amount of parallelism that needs to be exploited, organization of the processing blocks, etc.) that can lead to an efficient implementation of the edge-directed stereo algorithm on reconfigurable hardware. The resulting architecture was implemented on a Virtex-5 FPGA and evaluated in terms of processing speed, disparity map accuracy, and hardware overheads, against a stand-alone SAD-based architecture and existing hardware stereo matching systems.

# CHAPTER 4

# A Segmentation-Based Stereo Matching Hardware Design with Adaptive Support Weights: Balancing Speed and Accuracy in Embedded Vision Applications

*THIS chapter focuses on the implementation of a hardware stereo matching design that aims to improve the robustness of the matching process by incorporating image segmentation information and adaptive support weight aggregation. The chapter also presents a hardware adaptation methodology that consists of a series of hardware-oriented arithmetic approximations and optimization techniques, aiming to make the algorithm hardware-friendly and compatible with embedded constraints. A prototype of the architecture implemented as part of a complete stereo vision system on a high-end FPGA board featuring HDMI video support, is also presented, along with comparisons with related implementations in terms of various performance metrics (processing speed, matching accuracy and hardware overheads).*

## 4.1 Introduction - Motivation

Emerging applications of stereo vision such as robot navigation, space and avionics, and obstacle detection for autonomous vehicles, require real-time processing, low power consumption and accurate depth perception. Although the edge-directed disparity estimation architecture presented in Chapter 3 is compatible with embedded constraints, the implemented algorithm relies on a simple cost aggregation strategy with a fixed and rectangular correlation window. As such, it has difficulty in determining the best window shape and size for each pixel during the matching process, resulting in significant ambiguity along depth borders and areas with low texture.

Recent local stereo matching algorithms use adaptive weights during cost aggregation in an attempt to alleviate the aforementioned problems. They work by assigning different weights to the pixels in the support window, based on the importance of each pixel to the correlation process. There are different ways that can be utilized to determine the importance of each pixel. One way is to consider the color similarity of pixels, and assign larger weights to pixels having similar color with the central pixel of the window. Another approach is to assign weights based on the proximity distance of each pixel to the central one, using the Euclidean distance of the pixel coordinates. It is also possible to extract the weights by using image segmentation information, thus considering the consecutiveness of pixels and shape of segments. This latter method is based on the assumption that pixels lying in the same segment should be located at the same disparity.

ADSW algorithms are capable to deliver matching accuracy that is on par with global approaches [**129**]. However, this advantage does not come at free, since ADSW algorithms are usually more computationally expensive compared to fixed support algorithms, mainly due to the weight generation process that is performed exhaustively on a pixel-to-pixel basis. As such, their computational, and consequently hardware complexity, increases proportionally with the window size. In addition, these algorithms usually involve calculations (e.g. divisions, exponential functions, etc.) that are difficult to be implemented in hardware in a resource-efficient manner. Hence, a straight algorithm-to-hardware mapping that does not take into consideration any hardware-oriented algorithmic modifications/optimizations might not lead a real-time hardware design.

Being aware of the need for high quality, real-time disparity estimation in embedded vision applications, this Chapter investigates how to design a stereo matching architecture that utilizes ADSW aggregation, thus providing high matching accuracy, while at the same time, is able to provide real-time computation. The presented architecture is the first to utilize image segmentation information within the ADSW cost function, in an attempt to increase the robustness of the matching process. The Chapter also presents hardware-oriented algorithmic modifications and optimization techniques that make the algorithm hardware-friendly and suitable for efficient dedicated hardware implementation. The proposed architecture is able to provide an effective tradeoff between processing speed and matching quality, thus it can be used in embedded vision applications that need to provide high-quality disparity estimation in real time. Especially, given that the architecture obtains improved accuracy at depth discontinuity points, thanks to the advanced segmentation based cost aggregation strategy, it might be more suitable for tasks such as recognition or robotic grasping, where it is particularly important to achieve good accuracy near such points.

In summary, the main contributions of the segmentation-driven stereo matching architecture presented in this chapter, in relation to existing implementations reviewed earlier in Chapter 2.8, are as follows:

- It presents a dedicated hardware implementation of an ADSW stereo matching algorithm that integrates image segmentation information within the weight cost function to increase the robustness of the matching process.

- It introduces hardware-directed optimization techniques to adapt the algorithm for efficient hardware implementation, while keeping a balance between processing speed, accuracy and hardware efficiency.

- It presents the integration of a novel disparity refinement pipeline that benefits from the already available segmentation information to eliminate ambiguities and further smooth the resulting disparity maps.

- The presented architecture achieves an effective speed-accuracy tradeoff compared to existing state-of-the-art implementations.

The rest of the chapter is organized as follows. Section 4.2 presents the basic steps that constitute the implemented stereo matching algorithm. Section 4.3 introduces the hardware-oriented algorithmic modifications and optimization techniques. The proposed hardware architecture is described in Section 4.4, while Section 4.5 details the experimental platform and evaluation methodology for the architecture, along with performance results and discussion.

## 4.2 Segment-based adaptive support weight algorithm

The stereo correspondence algorithm implemented by the proposed disparity estimation hardware architecture is inspired by the ADSW algorithm proposed in [**63**]. This algorithm utilizes segmentation information within the weight cost function in order to increase the robustness of the matching process. The proposed architecture implements a modified version of the original algorithm, by introducing modifications of the core matching algorithm (e.g. a different matching cost, post-processing steps, etc) that aim to improve the overall accuracy. Furthermore, the architecture integrates hardware-oriented arithmetic approximations and optimization techniques in an attempt to make the algorithm hardware-friendly and compatible with embedded constraints. The major steps of the modified algorithm are described in this sections, while the hardware-oriented algorithmic modifications and optimizations are presented in Section 4.3.

The algorithm consists of four major steps: matching cost initialization, cost aggregation, disparity computation, and refinement, which reflect the taxonomy proposed in [**25**]. The overall flow of the algorithm is illustrated in Figure 4.1, while a detailed description of each individual step is given next.

## 4.2.1 Matching Cost Initialization

This step calculates a matching cost for each pixel $p$ at all possible disparities in the range $d_m$ to $d_M$. The algorithm proposed in [**63**] utilizes the truncated absolute difference (TAD) correlation metric (4.1) for the computation of a matching cost. In (4.1), $C_{TAD}(p,d)$ is the matching cost of pixel $p$ at disparity $d$, $I_r$ and $I_t$ are the pixel intensity values of the reference and target images, respectively, while $T_h$ is a truncation threshold.

$$C_{TAD}(p,d) = min(|I_r(x,y) - I_t(x+d,y)|, T_h), \quad T_h: truncation\ threshold \tag{4.1}$$

Existing literature [**59**] suggests that non-parametric costs like Rank and Census can have better and more robust overall accuracy compared to absolute and square differences (or their truncated versions), especially for images with radiometric differences. As such, the algorithm implemented by the proposed hardware architecture uses Census-based correlation in an attempt to make the matching process more robust. The Census transform is based on the relative ordering of pixel intensity values in a local neighborhood rather than the pixel values themselves, and therefore exhibits high resistance to radiometric distortion, vignette and noise [**59**]. Census transform encodes a neighborhood window $W_c$ surrounding a pixel $p$ to a bit vector of length $s_c^2$, where $s_c$ is the width of the Census window. This is done by applying an ordered set of comparisons of pixel intensities in $W_c$ in order to find out pixels that have lower intensity than the intensity of the central pixel $p$. If the intensity value of a neighboring pixel is

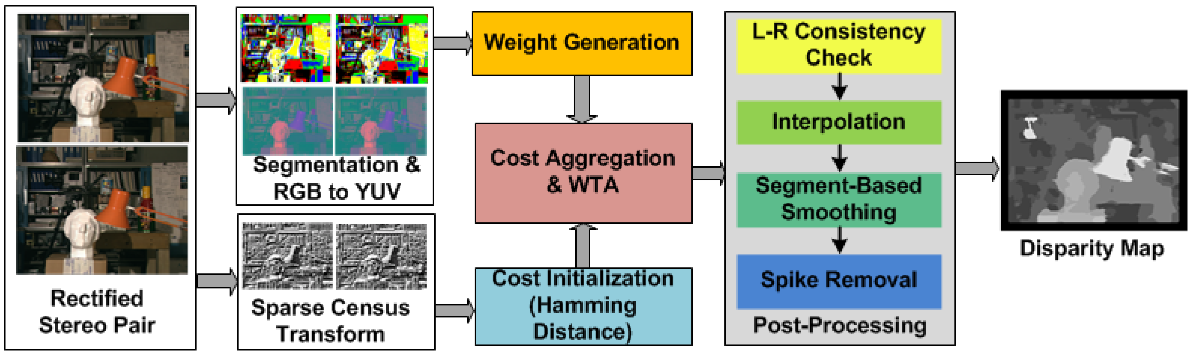


**Figure 4.1: Steps involved in the algorithm implemented by the proposed disparity estimation architecture.**

smaller than the intensity of $p$, then the corresponding position in the Census vector is set to l, otherwise is set to 0. The matching costs are then computed by the hamming distance of bit vectors at each disparity level as in (4.2), where $I_{r_{census}}$ and $I_{t_{census}}$ are the Census transformed images calculated by applying the Census transform over $I_r$ and $I_t$, respectively.

$$C_{census}\ (p,d) = Hamming(I_{r_{census}}\ (x,y), I_{t_{census}}\ (x+d,y)) \tag{4.2}$$

### *4.2.2* Cost Aggregation

This step aggregates the initial pixel-wise matching costs (calculated in the previous step) over a local support region $W_x$ around each pixel $p$. To ensure aggregation of only those pixels that lie at the same disparity with $p$, each point in $W_x$ is weighted based on the basis of its spatial and color distance with regards to $p$, as well as on information extracted from image segmentation. The weights are generated by taking into account both the reference and target images (symmetrical aggregation). Hence, the ADSW aggregation strategy generates a support-weight for each pixel that falls into the support window $W_r$ in the reference image, and correspondingly, in the support window $W_t$ in the target image. Let $p_c$ and $q_c$ being the central points of $W_r$ and $W_t$, respectively, then each matching cost (computed as the hamming distance of the Census bit vectors) for any point $p_i$ in $W_r$ corresponding to $q_i$ in $W_t$ is weighted by a coefficient $w'_r(p_c,p_i)$ and a coefficient $w'_t(q_c,q_i)$, defined as in (4.3).

$$w'_{r,t} = \begin{cases} 1.0 & p_i \in Seg_c \\ exp\left(-\frac{d_p(p_i,p_c)}{\gamma_p} - \frac{d_c\left(I_{r,t}(p_i), I_{r,t}(p_c)\right)}{\gamma_c}\right) & otherwise \end{cases} \tag{4.3}$$

where $Seg_c$ is the segment where the central point $p_c$ (or $q_c$) of the support window $W_r$ (or $W_t$) lies, $d_p$ and $d_c$ are the spatial and color distance between two coordinate pairs and two triplets in a chromatic color space, respectively, and $\gamma_p$ and $\gamma_c$ are two parameters of the algorithm. The final aggregated cost is then computed by summing up all the weighted pointwise scores, and normalizing by the weights sum as in (4.4).

$$C_{x+d,y} = \frac{\sum_{p_i \epsilon W_r, q_i \epsilon W_t} w'_r(p_c,p_i) \cdot w'_t(q_c,q_i) \cdot Hamming(p_i,q_i)}{\sum_{p_i \epsilon W_r, q_i \epsilon W_t} w'_r(p_c,p_i) \cdot w'_t(q_c,q_i)}\ ,\quad d \in [d_m,\ d_M] \tag{4.4}$$

### 4.2.3 Disparity Computation

The fact that ADSW methods belong to local stereo matching algorithms renders the disparity computation step simple. Once the aggregated costs are estimated for all disparity levels, the best disparity for the pixel $p$ is found by locating the disparity with the absolute minimum aggregated cost through the simple and commonly used Winner-Takes-All (WTA) approach. The WTA approach has a low complexity, and hence leads to an efficient hardware implementation without any algorithmic modifications required.

### 4.2.4 Disparity Refinement

This step eliminates ambiguous/inaccurate disparities, and involves further processing of the disparity map generated in the previous steps through left-to-right consistency check (L-R check), interpolation of occluded and mismatched regions, smoothing and spike removal. The L-R check (or mutual consistency) is applied on the left disparity map, and marks occluded or mismatched areas that usually arise due to lack of texture or camera noise as invalid (set to zero value). This is achieved by computing the left disparity map, $D_L$, (left image used as reference) as well as the right disparity map, $D_R$, (right image used as reference), and then finding where the two disparity maps are not negative of each other. The pixels marked as invalid by the L-R check are then given a new disparity value through median interpolation in a small neighborhood (of size $s_{mi} x s_{mi}$) around them, and around their corresponding pixels in the not consistency checked disparity map. The final interpolated pixel is the smallest between the two median values. However, only pixels that lie in the same segment with the central pixel of the window are taken into account during computing the median values. Therefore, this step is a weighted median filter with the weights being generated by utilizing the segmentation information already available from the cost aggregation step; the weight of a pixel is 1 if that pixel and the central one belong to the same segment, and 0 otherwise. The window size used in this step is selected to be small, as invalid pixels correspond to occlusions, which mostly belong to the local backgrounds. The next step smoothes the disparity map based on segmentation information (segment-based smoothing step). Smoothing is performed by median filtering over a neighborhood region of size $s_{sbs} x s_{sbs}$ in a similar way as discussed in the interpolation of the invalid pixels. However, this step usually involves a larger window size. Finally, a simple median filtering is applied over a window of size $s_{sr} x s_{sr}$ to remove remaining irregularities/spikes, and to further smooth the disparity map (spike removal step).

## 4.3 Hardware Adaptation Methodology

While the ADSW algorithm described in Section 4.1 leads to high disparity map accuracy, it is also a complex algorithm involving a lot of hardware demanding operations (e.g. multiplication, square root and division operations with floating point numbers). Moreover, unlike simple fixed support algorithms, it cannot take advantage of the “sliding window” technique [**64**], since the adaptive weights have to be recomputed at every pixel. Thus, cost aggregation needs to be performed in an exhaustive manner, making the hardware complexity directly dependent on the support window size. Therefore, the algorithm cannot be implemented efficiently in hardware just by following a straight algorithm-to-hardware mapping methodology. Instead, it should be first adapted by applying some arithmetic approximations and hardware-directed algorithmic modifications and optimizations techniques, directed towards an efficient hardware implementation. This section details the algorithm adaptation / hardware reduction methodology that we followed. The selection of the approximations is guided by the need to provide a tradeoff between processing performance, accuracy and hardware resource utilization.

The cost initialization step involves comparison operations for the transformation of the input images to images with Census vectors instead of pixel intensities, as well as hamming distance computations. While the Census transform is efficient with respect to hardware implementation, the size of the resulting census vectors ($s_c^2$) affects in a significant way both the accuracy of the matching process, as well as the hardware utilization of the hamming distance computation and cost aggregation circuits. Since Census transform is a type of window-based operation, the size of the local neighborhood plays an important role in the accuracy of the matching costs. As the Census window size increases, higher matching accuracy can be achieved [**65**]. However, an increased window size also causes object boundaries to be blurred and small objects to be removed for extensively large sizes [**106**]. As a result, there is always a tradeoff in selecting a good Census window size. This is also attributed by the fact that, as the size of the window increases, so do the hardware requirements; larger Census windows are associated with longer Census bit vectors per each pixel, thus making the hardware (especially of the cost aggregation step) more complex. In this work, we aimed to keep a balance between hardware efficiency and disparity map accuracy, thus we adopted a sparse Census transform. This involves sub-sampling the Census window with a sampling ratio of 2 in both directions (see Figure 4.2), thus reducing the hardware requirements of the stereo matching algorithm by a factor of 4; correlation is performed on shorter Census vectors. The basic idea behind this approach is that large sparse Census windows perform better (in terms of matching accuracy) than small dense windows

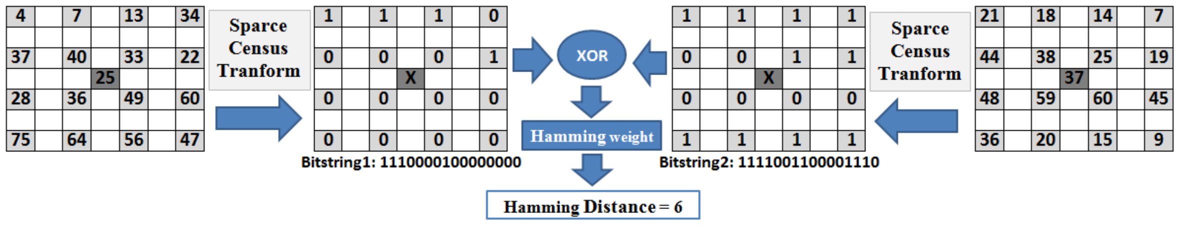


**Figure 4.2: Sparse Census transform and matching cost computation. The Hamming distance of the bitstrings resulting from sparse Census transform over $I_r$ and $I_t$ is the Hamming weight of Bitstring1 xor Bitstring2.**

with the same Census vector size, while requiring equal hardware resources. An example of a sparse Census transform and matching cost computation based on the resulting Census vectors is illustrated in Figure 4.2.

Correspondingly, the accuracy and hardware demands of the matching process are closely related to the size of the support window used in the cost aggregation step. Small windows do not contain enough information and lead to noisy results. On the other hand, aggregating over a large support window in ADSW methods was found to achieve better quality results, especially in noisy datasets [**65**]. This is a consequence of the fact that pixels inside the support window have different influence in the aggregation process, hence considerably reducing the edge-fattening problem when using large window sizes [**65**]. However, large windows are prohibitive for efficient dedicated hardware implementation, both in terms of on-chip memory requirements and computational resources. Although simplified adaptive weight techniques with 1D aggregation, or two-pass horizontal and vertical aggregation have been proposed [**130**] for efficient hardware implementation, we found that by sub-sampling the support window every *k* pixels in both directions (k=4 in the proposed hardware implementation) yields better quality. Therefore, we instead focus on this approach, which requires *k* times less hardware resources compared to the case of aggregating over the entire window. The motivation behind this approach is similar to the idea of the sparse Census transform. That is, large sparse support windows perform better than small dense support windows, while requiring the same amount of hardware resources.

The cost aggregation step is inherently the most computationally demanding step, as implied by equations (4.3) - (4.4). As such, besides sub-sampling the support window, we have adopted some hardware-directed arithmetic approximations to make the corresponding equations hardware-friendly. Firstly, we have chosen a simple image segmentation algorithm instead of utilizing the mean shift segmentation used in [**63**], as the computational complexity and memory requirements of mean shift segmentation make it unsuitable for embedded real-time applications. The k-means algorithm is simpler and can be implemented in hardware more efficiently. This work integrates an even simpler method, which partitions the image into segments using thresholding; we adopt this instead of the k-means, based on our observation that it has no negative impact on the overall disparity map accuracy.

Additionally, we eliminate the use of the spatial distance during the weight computation step based on our observation that it affects the accuracy of the disparity maps slightly. We also adopt YUV instead of CIELAB color representation (used in [**63**]) in the weight generation step. This allows the use of unsigned integers instead of signed floating-point integers, which are complex and difficult to be implemented efficiently in hardware. We also adopt Manhattan rather than Euclidean distance during the computation of the color distance between two YUV triplets. In this way, the square and square root operations are replaced by simple absolute difference and addition operations. Furthermore, the *exp(-x)* function is approximated by the $\lfloor 2^{8-x} \rfloor$ function, which assigns a maximum weight of 256 if the color distance is zero and a weight of 0 if the color distance is greater than 8. This function simplifies the circuits that implement the multiplication of the weight coefficients with the matching costs, as multiplications are reduced to left shift operations. The cost function is further simplified by setting $\gamma_c$ to 32 instead of 22 (the value used in [**63**]). This converts the division to a right shift operation. Lastly, the denominator of (4) is approximated by the nearest power of 2, allowing the division to be replaced by a right shift operation.

The majority of the post-processing steps utilize median filtering operations, which have been widely implemented in hardware using sorting networks (e.g. based on bubble sorting grid). However, the hardware requirements and complexity of this approach increase with respect to the window size. In this work, we have adopted the median filtering approach proposed in [**131**], which is based on cumulative histograms. This method can have a high degree of parallelism in hardware, and therefore it is more attractive compared to sorting network methods.

## 4.4 Proposed disparity estimation hardware architecture

The proposed disparity estimation architecture is decomposed into three major hardware units: the *Input Management Unit (IMU)*, the *Disparity Calculation Unit (DCU)* and the *Disparity Refinement Unit (DRU)*. These hardware units are fully pipelined so that to obtain best processing throughput. Source pixels come progressively in scanline order from the input port (stereo camera or external memory), enter the processing pipeline consisting of the three major units mentioned above, and after a pipeline latency, refined disparities are forwarded successively in scanline order to the output port, synchronized with the input pixel rate. The IMU performs RGB to YUV color space conversion, sparse Census transform and image segmentation on the incoming pixels. The DCU performs the correlation process, which is set up by a cascade of processes calculating the hamming distances, weight coefficients, aggregated costs and best scores through the winner-takes-all approach. The DCU generates the left and right disparity maps, which are then forwarded to the DRU to perform the L-R consistency check, and also to further refine the disparity values by interpolation of the mismatched pixels, segment-based smoothing and spike removal. Figure 4.3 shows a block diagram of the proposed architecture and the data flow between units.

### 4.4.1 Input Management Unit (IMU)

The IMU performs RGB to grayscale conversion, RGB to YUV color conversion, sparse Census transform and image segmentation on the left and right images, independently. It receives RGB color pixel values from the input port in a raster scan fashion, and converts them to grayscale and to their corresponding 8-bit YUV representation by utilizing two RGB to grayscale converters (rgb2gray) and two RGB to YUV color space converters (rgb2yuv), respectively. The resulting grayscale and YUV values are stored in line buffer memories (scanline buffers), which are employed to provide pipelined window context to the Census Transform and Segmentation units, through the use of register matrices (window buffers and

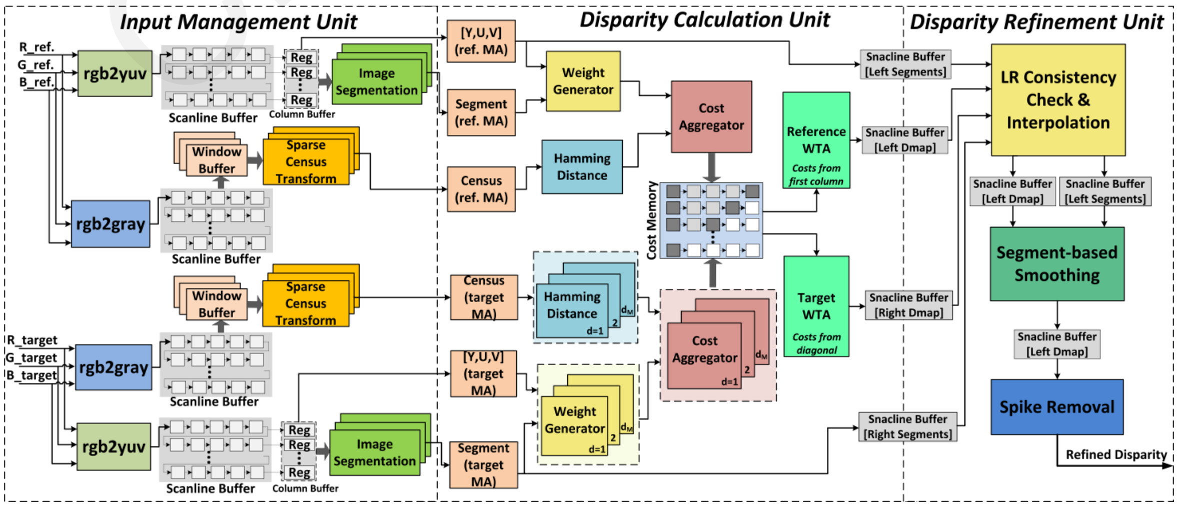


**Figure 4.3: Proposed segmentation-driven disparity estimation hardware architecture.**

column buffers). As grayscale and YUV values come in continuously to the scanline buffers, the $s_{sw} x s_{sw}$ support windows in $I_r$ and $I_t$ slide over the whole frames in a scanline order. The window and column buffers store all pixels required to compute the Census vectors and segments corresponding to the rightmost pixels of the sub-sampled support windows in $I_r$ and $I_t$. We used a 17x17 support window, sub-sampled every 4 pixels in each direction, resulting in a sub-sampled support window size of 5x5. Thus, the IMU utilizes a total of 10 sparse Census transform and segmentation units (5 per image).

The sparse Census transform units mainly involve comparison operations between the central pixels in the sparse Census windows and their surrounding pixels, in order to convert the grayscale values into Census vector values. We use a window size of 7x7 for the Census transform, sub-sampled every 2 pixels in both directions, thus resulting in a sparse Census window of 4x4 and a 16-bit long transformed bit vector per each pixel.

The image segmentation units receive $Y$ values from the column buffers and the number of segments $n$ given as input to the system (maximum supported $n$ is 32). A label (an unsigned integer in the range 1 to $n$) computed by a simple method that multiplies the input grayscale value by the value of $n/256$ is assigned to each input $Y$ value. The multiplication is performed using fixed-point arithmetic with 8-bits of integer and 16-bits of fraction. The values of $n/256$ for all possible values of $n$ are stored in a look-up table. The result is given by taking the 5 most significant bits of the multiplication operation.

### 4.4.2 Disparity Calculation Unit (DCU)

The DCU is provided with on-chip Memory Arrangements (MAs) that temporarily store the pixels required to perform correlation between the sub-sampled versions of $W_r$ in $I_r$ and the $d_M$ candidate support windows $W_t$ in $I_t$. In particular, it is provided with 5 MAs per input image, which store the Y, U, V color values, the Census vectors and the segments, received as input from the IMU. The DCU also consists of weight generation units, hamming distance computation units, units that compute the aggregated costs at different disparity levels, a matching cost memory and WTA units for the left and right disparity maps.

Each weight generator computes the weight coefficients $w'_{r,t}$ for a sub-sampled version of the support window $W_{r,t}$ in parallel. It receives the segment information and the YUV color values corresponding to the $s_{sw\_sub} x s_{sw\_sub}$ sub-sampled support window from the Segments and [Y,U,V] MAs, respectively, and computes the $s^2_{sw\_sub}$ weight coefficients using $s^2_{sw\_sub}$

instances of the circuit shown in Figure 4.4 (b). That circuit consists of a comparator, a Manhattan distance core and a weight table (LUT). Since the multiplication of the matching costs by the weight coefficients (cost aggregation step) is performed using shifters instead of multipliers, each location $x$ of the LUT stores the shift amount corresponding to the weight coefficient $\lfloor 2^{8-x} \rfloor$. This shift amount is equal to the binary logarithm of $\lfloor 2^{8-x} \rfloor$, except from values of $x$ greater than 8, for which a binary logarithm does not exist. In that special case, the corresponding entries in the LUT are set to a number, which is large enough so that the result of a shift operation by that number is equal to zero. The comparator determines whether the pixel at location $(i, j)$ in the sub-sampled support window lies in the same segment with the central pixel of the window. The result of the comparator specifies whether the shift amount that corresponds to the weight coefficient $w'_{r,t}$ will be assigned to the shift amount corresponding to the maximum weight (8 in our case) or whether it will be looked up in the LUT using the color distance generated by the Manhattan Distance core as index.

The DCU utilizes Hamming distance computation units (Figure 4.4 (a)) that compute the matching costs between corresponding pixels in the sub-sampled support windows stored in the Census MAs, using XOR circuits and adder trees for the Hamming weight computations. The weights and hamming distances are the inputs of the cost aggregators. The architecture of a cost aggregator unit is shown in Figure 4.4 (c). It shifts the hamming distances by the shift amounts corresponding to the weight coefficients $w'_r$ and $w'_t$ using a series of left shifters (equivalent to multiplying the costs by $w'_r$ and $w'_t$). The final aggregated cost is computed by

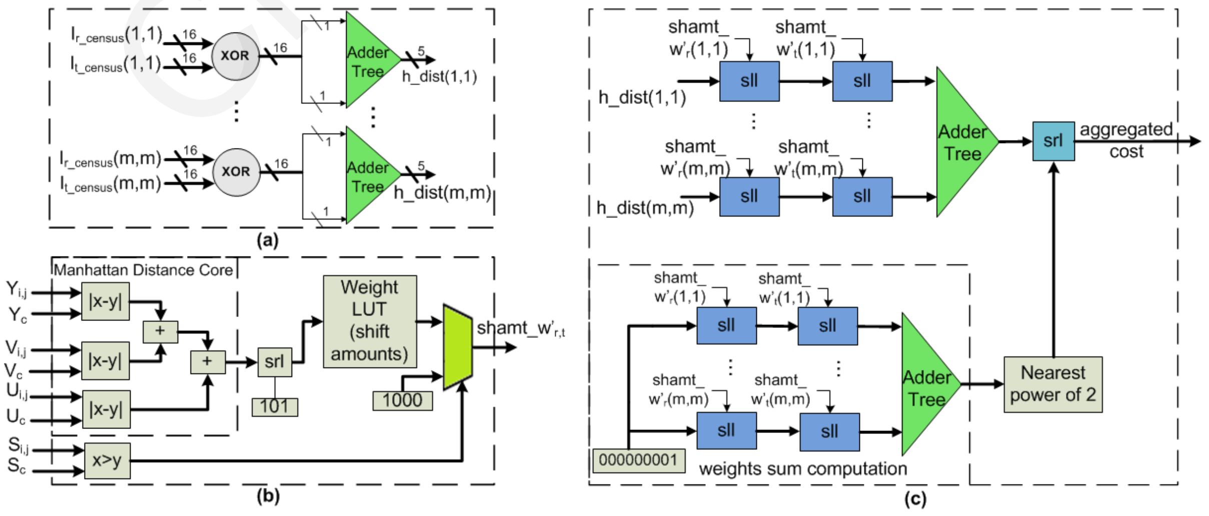


**Figure 4.4: Major units involved in the disparity calculation unit. (a) Hamming distance computation, (b) Weight generation, (c) Cost aggregation.**

summing the outputs of the left shifters using a tree adder, and then normalizing (dividing) it by the weights sum, which, before being used for division, is rounded to the nearest power of 2 by using tree comparators. This enables a cost-effective implementation of the division using a right shifter.

The aggregated costs for the left disparity map are stored in the cost memory and are reused to compute the aggregated costs for the right disparity map, as proposed in [**132**], [**89**]. These scores are then utilized to compute the left and right disparity maps by the reference and target WTA units, respectively. The only difference is that the reference WTA unit operates in a vertical direction in the cost memory to search the best L-R matches, while the target WTA unit operates in a diagonal direction. The WTA units are mainly composed of comparators organized in tree structures and pipelined to keep the delay in the critical path small.

### 4.4.3 Disparity Refinement Unit (DRU)

The processing pipeline of the DRU begins with checking disparity consistency between the left and right disparity maps. The left and right disparity values computed by the DCU are stored in scanline buffers along with their corresponding segments, and are used to identify occluded and mismatched correspondences in the left disparity map $D_L$ (used as reference) and correct them through interpolation. While processing pixel $L(x, y)$ in $D_L$, the scanline buffer that stores the disparity values from the right disparity map, $D_R$, outputs $d_M$ disparity values in the interval $[R(x - d_M, y)$ to $R(x, y)]$. Those values are stored temporally in an on-chip memory, which is addressed by the disparity value $D_L(x, y)$ at point $L(x, y)$. The two disparities are compared in order to decide whether the disparity passes the L-R check. If the L-R check is passed, the disparity value at $L(x, y)$ forms the output of the L-R Check & Interpolation unit. Otherwise, the mismatched pixel is interpolated by taking the minimum value between the median values in a window around $L(x, y)$ in $D_L$ and around $R(x - D_L(x, y),\ y)$ in $D_R$. The segmentation information from the scanline buffers is incorporated in the median filters so that to account only for those pixels that lie in the same segment with the central one. The refined left disparity map is then passed to the segment-based smoothing unit, which smoothes the disparity map by applying segmentation-directed median filtering on all disparity map points. Lastly, a simple median filtering process is applied to eliminate remaining spikes.

All post-processing units employed in the DRU utilize median filtering operations with different window sizes. To implement an efficient hardware implementation of the median filtering operations, which is also scalable to large window sizes, we followed the approach in [**131**], which is based on constructing cumulative histograms of the input values (disparities in our case). However, we extended this approach to the case of adaptive weighted filtering. The histogram-based median filter architecture is shown in Figure 4.5. For each disparity value in the window being processed, the corresponding bin and all subsequent bins are incremented. To implement this requires a register for each possible input disparity value, which implies that the number of bins is equal to the maximum disparity range $d_M$. Each disparity value in the window addresses a ROM (contents shown in Figure 4.5), whose output decides which bins to increment.

The entire architecture of the median filter has been designed in a parallel manner, allowing the computation of the median value for a window of size $mxm$ in a single clock cycle. This was achieved by having $m^2$ $d_M$-bit ROMs and an adder tree before each bin to add the number of times (1 to $m^2$) each bin should be incremented. The segmentation information is used to generate binary weights; 1 if the received value lies in the same segment with the central pixel, 0 otherwise. These binary weights are generated by a weigh generator circuit, mainly consisting of comparators, and are used as the enable signals of the ROMs. In the case of the spike removal unit, which does not incorporate segmentation information as part of the median filtering process, the enable signals are all set to one. The output median value is eventually calculated by finding the bin, whose count is the first to reach the value of $d_{M+1}$ or surpass it. To implement this, the context of each bin is compared to the value of $d_{M+1}$, giving 0 if the bin count is smaller, and 1 otherwise. The result of the comparator circuits will be zero

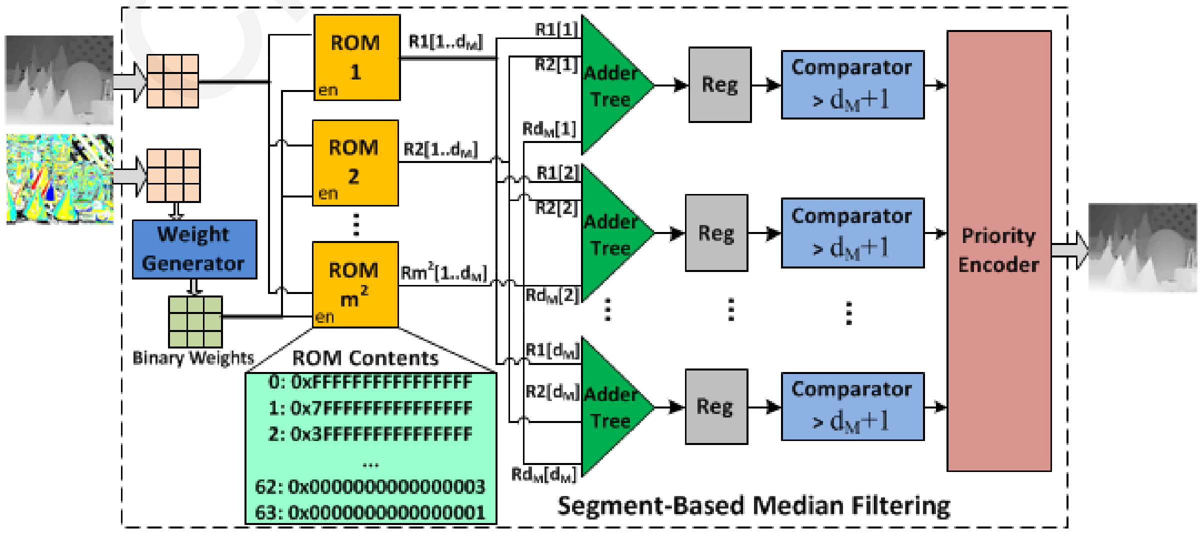


**Figure 4.5: Segment-based smoothing based on cumulative histograms.**

for all bins before the one containing the median value and all the others will be 1. A priority encoder is employed to isolate that bin.

## 4.5 Experimental methodology & Results

### 4.5.1 Experimental Platform & Synthesis Results

To evaluate the proposed architecture, we followed two major steps. First, we designed and verified the architecture using FPGA emulation. Second, we designed a full-custom ASIC implementation over a commercial CMOS library, in order give an insight about the differences in speed and power consumption when the proposed architecture is moved from the FPGA design point to the ASIC design point. The FPGA system was designed with emphasis on the corresponding hardware constraints (fixed amount of resources, fixed placement and routing), while the ASIC design has targeted more parallel configurations with large image sizes, in order to indicate the feasibility of the proposed architecture for large-scale embedded systems.

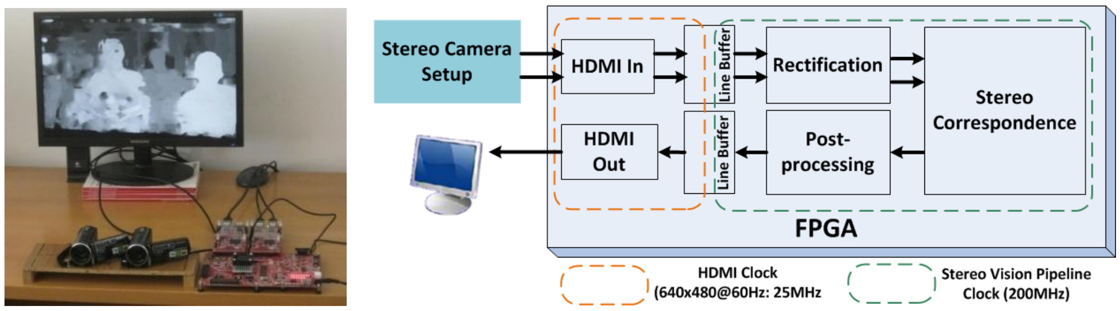
### 4.5.2 FPGA Implementation and Emulation

In order to evaluate the proposed system architecture and verify its operation, we have implemented it as part of a complete stereo vision system on the Inrevium Kintex-7 FPGA ACDC 1.0 Baseboard [**133**], which is equipped with a Xilinx Kintex-7 FPGA (XC7K325T-FFG900) [**134**]. We used a custom-built stereo camera system consisting of two Sony Handycam video cameras configured to output 640x480 images at 60 fps. The stereo camera system was calibrated using the Camera Calibration Toolbox for MATLAB [**47**], and was then used to capture stereo image pairs, which were rectified and used as input data to the system architecture shown in Fig. 3. We have implemented an LUT-based stereo image rectification hardware unit, which was used to determine displacements of pixels in the rectified images from their positions in the original images. To reduce the size of the rectification LUTs, we followed the approach in [**55**], where the LUTs are compressed based on differential encoding. That method resulted in a compression ratio of ~72.3%, thus allowing the rectification to be implemented with compact hardware resources. The cameras were interfaced to the FPGA board through HDMI interface, by using video interface specific FMC daughter cards (Inrevium HDMI 1.4a Tx and Rx) [**135**].

While the proposed disparity estimation architecture is scalable and can support parallel computation of all disparity levels in a single clock cycle, the FPGA implementation of the architecture was configured taking into account the available FPGA resources and frame-rate of the cameras. Configuration was done by changing a few configuration parameters of the VHDL description. In particular, the DCU was configured to cover a maximum disparity range of 64 levels, using eight cost aggregators working in parallel. Thus, the system implemented on the FPGA prototype can generate one disparity value every eight clock cycles.

Since the HDMI and the system architecture use a different clock source, the video sequences were temporarily stored in line buffers so as to ensure that the input stereo pairs were forwarded in a continuous fashion to the stereo vision pipeline that consisted of the rectification unit, the stereo correspondence unit (IMU & DCU) and post-processing modules. The refined disparity maps were also synchronized with the pixel clock through a line buffer, and directed to a standard TFT monitor through the TB-FMCH-HDMI2-TX card [**135**]. The system diagram of the prototype FPGA implementation is given in Figure 4.6.

The FGPA utilization reports from the Xilinx ISE Synthesis Tool are summarized in Table 4.1 (footnote lists the various system parameters). The mapping of the different system components on the FPGA fabric was performed taking into account the computational and memory demands of each component. We also mapped the different components in a way that maintains a good ratio between instantiated Slice Look-Up Tables (LUTs), Slice Registers and Block RAMs (BRAMs), in order to balance the utilization of the available resources. The entire system implemented on the FPGA utilizes ~53% of the available Slice Registers and ~66% of the available LUTs. It also utilizes ~5% of the DSP units and ~67% of the BRAM memories. The FPGA resources are dominated by the stereo correspondence unit (IMU & DCU), while the pre-processing (rectification) and post-processing modules require fewer resources. The system can operate at a maximum operating frequency of 200 MHz. With respect to the on-chip power consumption, the entire system dissipates 6.7 W; the power



(a) (b)

**Figure 4.6: FPGA prototype system. (a) Experimental testbed used to verify the operation of the system architecture, (b) Block diagram of system components.**

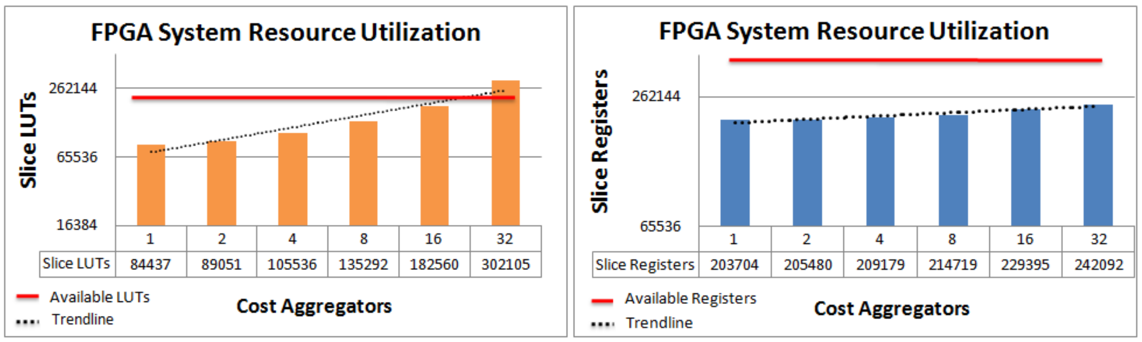
**Table 4.1: FPGA Prototype Hardware Overheads.**

| Design Unit | Slice Registers *(total=407600)* | Slice LUTs *(total=203800)* | DSP48E1s *(total=840)* | BRAM *(total=445)* | Freq. *(MHz)* |
|---|---|---|---|---|---|
| **Stereo Image Rectification** | 152 *(0.04%)* | 241 *(0.1%)* | 0 | 132 *(29.7%)* | 230 |
| **Stereo Matching (IMU & DCU)** | 109377 *(26.8%)* | 98799 *(48.5%)* | 40 - *(4.8%)* | 0 | 208 |
| **L-R Check & Interpolation** | 2035 *(0.5%)* | 3274 *(1.6%)* | 0 | 18 *(4.0%)* | 491 |
| **Segment-based Smoothing** | 84504 *(20.7%)* | 17274 *(8.5%)* | 0 | 121 *(27.2%)* | 495 |
| **Spike Removal** | 838 *(0.2%)* | 2699 *(1.3%)* | 0 | 25 *(5.6%)* | 498 |
| **Full System on Kintex-7 Board (including HDMI controllers)** | **214719** ***(52.7%)*** | **135292** ***(66.4%)*** | **40** ***(4.8%)*** | **296** ***(66.5%)*** | **200** |

* Parameters:$\{image\ size = 640x480,\ d_M = 64,\ s_c = 7,\ s_{c\_sub} = 4,\ s_{sw} = 17,\ s_{sw\_sub} = 5,\ s_{mi} = 5,\ s_{sbs} = 11,\ s_{sr} = 5\}$

consumed through the operation of the FPGA device (dynamic power) is ~5.9 W, while the power consumed regardless of design activity (static power) is ~0.8 W. These values indicate that the architecture implemented on the FPGA device is capable of realistic embedded system usage.

The scalability of the proposed disparity estimation architecture is explored by illustrating how the amount of utilized Slice LUTs and Slice Registers of the FPGA implementation (full system including rectification, stereo correspondence and post-processing) is increased with the number of cost aggregators (see Figure 4.7). The scalability figure shows that the implementation complexity of the complete system implemented on the FPGA scales nearly linearly with the number of aggregators. The figure also shows that the targeted FPGA can fit up to 16 aggregators. Though, the architecture was tailored to 8 aggregators, as this number was enough to support the frame rate of the stereo camera system (60 fps). Obviously, given cameras with higher rates, a fully-parallel implementation of the proposed architecture on a feature-rich FPGA platform that provides more processing power



| | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| Slice LUTs | 84437 | 89051 | 105536 | 135292 | 182560 | 302105 |

| | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| Slice Registers | 203704 | 205480 | 209179 | 214719 | 229395 | 242092 |

**Figure 4.7: Scalability analysis of full system on Kintex-7 FPGA (impact of the number of aggregators on the amount of utilized FPGA resources). Axes with logarithmic scale (base 2). Y-axes do not start at zero.**

(in the form of more reconfigurable logic, registers and embedded Block RAM) than the one experimented with in this work, would allow further exploitation of disparity-level parallelism, thus providing higher processing performance.

Table 4.2 provides FPGA resource utilization details of the referenced FPGA implementations listed in Table III; some works are omitted as they do not provide the corresponding results. The FPGA device used in each work influences the resource utilization considerably. The resources may vary between different FPGA vendors, FPGA technologies and FPGA families, as some FPGA devices use 4-input LUTs, while others use 6-input or 8-input LUTs. The resource utilization also depends on factors such as the effectiveness of the synthesis tool and the quality of the place and route software. Consequently, there is no direct way to compare the resource utilization of hardware designs implemented on different FPGA devices. However, Table II is provided as an indication of the hardware overheads incurred by each implementation; along with the other performance data (processing speed and matching accuracy) and implementation details provided in Section 6.3, we aim to provide a complete view of the different implementations based on comprehensive performance data.

### 4.5.3 Exploring large-scale scalability through an ASIC Implementation

Besides prototyping the proposed disparity estimation architecture on the Kintex-7 FPGA, we also evaluated its performance when implemented through a standard ASIC design flow targeting a commercial CMOS 65-nm cell library. The corresponding ASIC implementation was used to obtain experimental insights on the scalability and feasibility of the proposed architecture towards very large-scale integration, while optimizing the design in terms of area and power efficiency. In particular, we implemented three configurations of the

**Table 4.2: Resource Utilization Of Different FPGA Implementations.**

| | Logic Elements | | Embedded Multipliers | Block Memory (Kbits) | FPGA Platform | Frequency (MHz) |
|---|---|---|---|---|---|---|
| | LUTs | Registers | | | | |
| **Georgoulas [2009]** | 143653 | 15442 | n.a. | 0.1875 | Altera EP4SGX290 | 511 |
| **Diaz [2007]** | 26096 | 26096 | 59 | 1278 | Xilinx Virtex-II XC2V6000-4 | 65 |
| **Jin [2010]** | 60598 | 53616 | 12 | 5796 | Xilinx Virtex-5 XC4VLX200-10 | 93.1 |
| **Ding [2011]** | 50585 | 35020 | 2 | 1404 | Xilinx Virtex-5 LX155T | 60 |
| **Ambrosch [2010]** | 139606 | n.a.[2] | n.a. | 768 | Altera Stratix I | 133 |
| **Gehrig [2009]** | 30000 | n.a. | n.a. | 1200 | Xilinx Spartan-3a 3400 FPGA | 133 |
| **Banz [2011][1]** | 50144 | n.a. | 50 | 2790 | Xilinx Virtex-5 XC5LX220T | 61.2 |
| **Ttofis [2012]** | 62213 | 55594 | 16 | 1080 | Xilinx Virtex-5 LX110T | 155 |
| **Proposed Work** | 135292 | 214719 | 40 | 10656 | Xilinx Kintex-7 | 200 |

[1] The paper provides the resource utilization for different systolic array configurations. The results provided here refer to a systolic array with 5 parallel rows, and a number of parallel disparity levels of 2.
[2] Data not available.

**Table 4.3: ASIC implementation synthesis results.**

| | Configuration 1 | Configuration 2 | Configuration 3 |
|---|---|---|---|
| **Technology** | CMOS 65-nm | CMOS 65-nm | CMOS 65-nm |
| **Combinatorial Logic Gate Count Equivalent Estimate** | ~625,000 | ~1,850,000 | ~1,975,000 |
| **Memory Size Estimate** | ~886.2 kB | ~888.5 kB | ~970.8 kB |
| **Image Size** | 640x480 | 640x480 | 1280x1024 |
| **Disparity Range** | 64 | 64 | 64 |
| **Number of cost aggregators** | 8 | 64 | 64 |
| **Targeted Frequency** | 200 MHz | 200 MHz | 200 MHz |
| **Processing performance** | 80 fps - 1572 MDE/s | 575 fps - 11304 MDE/s | 143 fps - 11995 MDE/s |
| **Dynamic Power** | ~47 mW | ~246 mW | ~430 mW |

* Parameters: $\{s_c = 7, s_{c\_sub} = 4, s_{sw} = 17, s_{sw\_sub} = 5, s_{mi} = 5, s_{sbs} = 11, s_{sr} = 5\}$

proposed architecture: one similar with the FPGA implementation featuring 8 cost aggregators working in parallel (Configuration 1), and two fully parallel architectures targeting 640x480 and 1280x1024 images, respectively (Configurations 2 & 3). The different ASIC configurations were synthesized and implemented using Synopsys Design Compiler, with 1V power supply voltage and a targeted frequency of 200MHz. It must be noted that the targeted frequency was selected to give an indicative performance when compared to the FPGA implementation; the reported slack times from the synthesis results indicate that a much higher frequency is feasible, as the design is highly pipelined and simple due to the proposed optimizations. Table 4.3 lists the core characteristics of the synthesized designs. Dynamic power results were obtained using a 50% switching activity; dynamic power consumption for computing a full stereo pair of images (of the corresponding sizes) is also listed in Table 4.3. The ASIC synthesis results indicate that the proposed architecture is capable to achieve real-time frame rates even at high resolution images, while consuming very little power.

### 4.5.4 Performance Results and Discussion

#### 4.5.4.1 *Processing Speed*

We measured the processing speed of the proposed system architecture in terms of the frames processed per second (fps) using synthetic stereo image pairs from [**25**] as benchmarks. A real-time performance of around 30 fps is sufficient for most video-rate stereo vision applications. However, applications with multiple data streams (e.g. multi-view stereo vision) and high-resolution video analysis may require higher frame rates. Through the computational reduction obtained by the hardware-directed optimization techniques discussed in Section 4.2, and the highly parallel and pipelined structure of the proposed system architecture, both the FPGA and ASIC implementations are capable of high processing performance for 640x480 images (60 fps and 575 fps, respectively) and a disparity range of 64 pixels. While the FPGA implementation was tailored to the available FPGA resources and camera frame-rate, and is capable to generate one disparity value every eight cycles, the ASIC implementation has been designed in a fully parallel manner and generates one disparity every cycle. It is also capable to achieve real-time performance for larger image sizes (143 fps for 1280x1024 images). Such rates are significant considering that the proposed architecture implements one of the most complex and accurate local stereo correspondence algorithms available in literature [**61**].

Table 4.4 presents a comparison between existing implementations and the proposed architecture for both the FPGA and ASIC implementations. Performance is provided in frames per seconds (fps) and also in Million Disparity Estimations per second (MDE/s); the latter reflects both the density and the speed of the system (number of pixels x disparity range x frame rate). We adopted these metrics, since they are the two most common metrics used to evaluate the throughput of stereo vision algorithms and architectures in literature. However, we acknowledge that a quantitative performance comparison of the reference implementations included in Table 4.4 is not a simple problem, as it should also take into account multiple other factors that influence the processing performance of each implementation. Such factors include the complexity of the adopted algorithm, the amount of parallelism employed, as well as several hardware-specific details (e.g. implementation platform, operating frequency, available memory, etc). Baring this in mind, we also report available algorithmic and hardware-specific details of the reference implementations (in columns 6-7 of Table 4.4) to allow the reader to make such comparisons.

As shown in Table 4.4, the proposed system (both FPGA and ASIC versions) obtained a MDE/s performance, which is among the highest reported in the table, outperforming the implementations based on CPU, Cell, DSP and GPU platforms, as well as the dedicated hardware implementations of fixed support methods in [**107**], [**109**]. Furthermore, the processing performance obtained by the proposed architecture is higher when compared to the FPGA-based implementations of SGM algorithms in [**110**], [**113**] and similar to the SGM-based implementation in [**112**]. Only the implementation in [**105**] is faster than both the proposed FPGA and ASIC implementations, while the works in [**126**], [**108**] outperform only the FPGA-based implementation of the proposed architecture. It must be noted that the systems in [**105**], [**108**] use rectangular and fixed support regions during cost aggregation. With carefully designed data reuse techniques (e.g. “sliding window” technique [**64**]) and by taking advantage of the incremental calculations, such systems can fully exploit the

Table 4.4: Summary of existing disparity estimation systems.

| Work | Image Size | D | MDE/s* | fps | Method | Platform |
|---|---|---|---|---|---|---|
| Hirschmuller [2004] | 320x240 | 32 | 11.5 | 4.7 | Multiple window | Pentium II, 450 MHz |
| Forstmann [2008] | 384x288 | 32 | 143.3 | 40.5 | Global based on DP | AMD XP 2800+ 2.2GHz |
| Zinner [2008] | 320x240 | 30 | 96.8 | 42 | Census-based | Intel Core 2 Duo 2GHz |
| Wang [2006] | 320x240 | 16 | 52.8 | 43 | ADSW and DP | 3 GHz PC & ATI Radeon XL1800 |
| Baha [2010] | 384x288 | 16 | 8.8 | 5 | Variable Support Window | 2.2 GHz Core Duo |
| Gehrig [2010] | 640x320 | 16 | 72.7 | 22.2 | SGM | Intel Core i7-975ex 3GHz |
| Yang [2006] | 384x288 | 16 | 22.2 | 12.77 | Hierarchical BP | NVIDIA Geforce 7900 GTX |
| Yang [2005] | 512x512 | 92 | 289 | 13.8 | Block-matching (SAD) | ATI Radeon 9800 |
| Xiang [2012] | 384x288 | 16 | 18.8 | 10.6 | Global belief propagation | NVIDIA Geforce GTX 260 |
| Ernst [2008] | 320x240 | 64 | 63.9 | 13 | Mutual information SGM | NVIDIA GeForce 8800 ULTRA |
| Konolige [1997] | 160x120 | 64 | 9.8 | 8 | LOG transform | ADSP 2181, 33MHz |
| Berretty [2007] | 720x480 | 80 | 829 | 30 | Block-matching (SAD) | TM3270 embedded media proc. |
| Khaleghi [2008] | 160x120 | n.a | n.a | 20 | Hamming distance correl. | ADSP-BF561, 600MHz |
| Chang [2007] | 384x288 | 16 | 88.5 | 50 | Jigsaw matching | TMS320C6414T-1000 |
| Humenberger [2010] | 320x240 | 15 | 660.9 | 573.7 | Optimized Census Transform | GeForce GTX 280 |
| | | | 30.41 | 26.4 | | DSP 1GHz T1 TMS320C6416 |
| | | | 72.5 | 62.9 | | PC Intel Core2 Duo 2 GHz |
| Cavanag | 450x375 | 59 | 1.39 | 0.14 | ADSW | Cell Processor on PlayStation 3 |
| Liu [2009] | 384x288 | 16 | 0.9 | 0.5 | Dynamic programming | Cell Processor on PlayStation 3 |
| Hsieh [2012] | 384x288 | 16 | 11.5 | 6.5 | Belief propagation | Cell Processor on PlayStation 3 |
| Georgoulas [2009] | 800x600 | 80 | 21120 | 550 | Block-matching (SAD) | EP4SGX290 (511MHz) |
| Diaz [2007] | 1280x960 | 29 | 1885 | 52 | Phase based | Xilinx Virtex-II (65MHz) |
| Darabiha [2006] | 256x360 | 20 | 55.2 | 30.3 | Phase correlation | Xilinx Virtex-4 2000E FPGA |
| Jin [2010] | 640x480 | 64 | 4522 | 230 | Census transform | Virtex-5 XC4VLX200-10 (93.1 MHz) |
| Yen-Chang [2010] | 352x288 | 64 | 272.5 | 42 | Mini-census ADSW | UMC 90ns Std. Cell |
| Ding [2011] | 640x480 | 60 | 940 | 51 | TSAD Adaptive Weights | Virtex-5 LX155T |
| Ambrosch [2010] | 750x400 | 60 | 1080 | 60 | SAD-IGMCT | Altera Stratix I (133MHz) |
| Gehrig [2009] | 340x200 | 64 | 117.5 | 27 | ZSAD SGM | Spartan-3 FPGA |
| Banz [2011] | 640x480 | 128 | 1179 | 30 | SGM | Xilinx Virtex-5 (12MHz – 208 MHz) |
| Hirschmuller [2012] | 640x480 | 64 | 328.3 | 16.7 | Census SGM | Xilinx Virtex-5 (125 MHz) |
| Ttofis [2012] | 640x480 | 64 | 589 | 30 | Segment-based ADSW | Virtex-5 LX110T FPGA (155 MHz) |
| Proposed Work | 640x480 | 64 | 1179 | 60 | Segment-based ADSW | Kintex-7 FPGA (200 MHz) |
| | | | 11304 | 575 | | CMOS 65-nm cell library (200 MHz) |

* MDE/s = (Image size x Disparity range x Frames per second) / 1,000,000

parallelism and achieve very high speeds. However, due to the simplicity of the adopted algorithms (and their associated problems in determining a proper window shape and size during aggregation), the implemented systems in [**105**], [**108**] do not lead to high disparity map accuracy. Although the proposed system is slower compared to the aforementioned state-of-the-art implementations of fixed support methods (due to the inherent complexity of the ADSW algorithm adopted in this work), it generates much better accuracy (Table 4.6), thanks to the advanced segment-based ADSW cost aggregation strategy. Finally, it must be noted that a direct comparison with the system in [**126**] is not possible, as the corresponding paper does not provide quality results.

When compared to hardware implementations of ADSW algorithms [**114**], [**115**], both the FPGA and ASIC versions of the proposed architecture show better processing performance. Although the complexity of the segment-based ADSW algorithm implemented by the proposed architecture is twice as large as the complexity of the ADSW algorithms in [**114**], [**115**] (as shown in [**61**] by measuring the time needed to process a reference stereo pair on the same machine), the proposed FPGA and ASIC systems outperform these implementations, indicating the potential use of the architecture in high-performance embedded applications that demand accurate disparity maps.

### 4.5.4.2 *Disparity Map Quality Analysis*

The quality of the disparity maps generated by the proposed hardware architecture was evaluated quantitatively on a set of benchmark stereo images from the Middlebury dataset [**25**], and by measuring the incorrect disparity estimates using the percentage of bad matching pixels (5), a commonly accepted metric [**25**]. This metric measures the quality based on known ground truth disparity maps at three different kind of regions, namely, *nonocc* (all points except for occluded areas), *all* (all points including half-occluded regions), and *disc* (only points along depth discontinuities). In equation (4.5), the subscript $R$ identifies the set of points (region) over which the percentage of bad matching pixels is measured ( i.e. $R \in$ *{nonocc, all, disc}* ), $d_c(x,y)$ and $d_t(x,y)$ are the computed and ground truth disparity maps, respectively, $N_R$ is the total number of pixels in the region of interest, and $\delta_d$ is the disparity error tolerance. We set $\delta_d = 1.0$ for all experiments we carried out, as this value coincides with some previously published works.

$$B_R = \frac{1}{N_R} \sum_{(x,y)\in R} (|d_c(x,y)| - |d_t(x,y)| < \delta_d) \tag{4.5}$$

We measured the percentage of bad matching pixels using four reference images on the dataset (Tsukuba, Venus, Teddy and Cones). Evaluation results are shown in Figure 4.8 (a), and are also given in Table 4.5. Furthermore, the average percentage of bad matching pixels over the four test images is given for each different region independently in Table 4.6 (columns 2-4). Based on these averages, the overall percentage of bad matching pixels is also extracted (*avg* term in column 1 of Table 4.6). Subsequently, an accuracy (acc) term stating the ratio of the pixels given a correct disparity value (as compared with the ground truth) can also be derived (acc = 100-avg). The proposed architecture obtains 3.99% bad matching pixels in non-occlusion regions, 5.00% in the whole disparity map and 16.5% in discontinuity regions, resulting in an overall percentage of bad matching pixels of 9.79%, which corresponds to an overall accuracy of 90.21%.

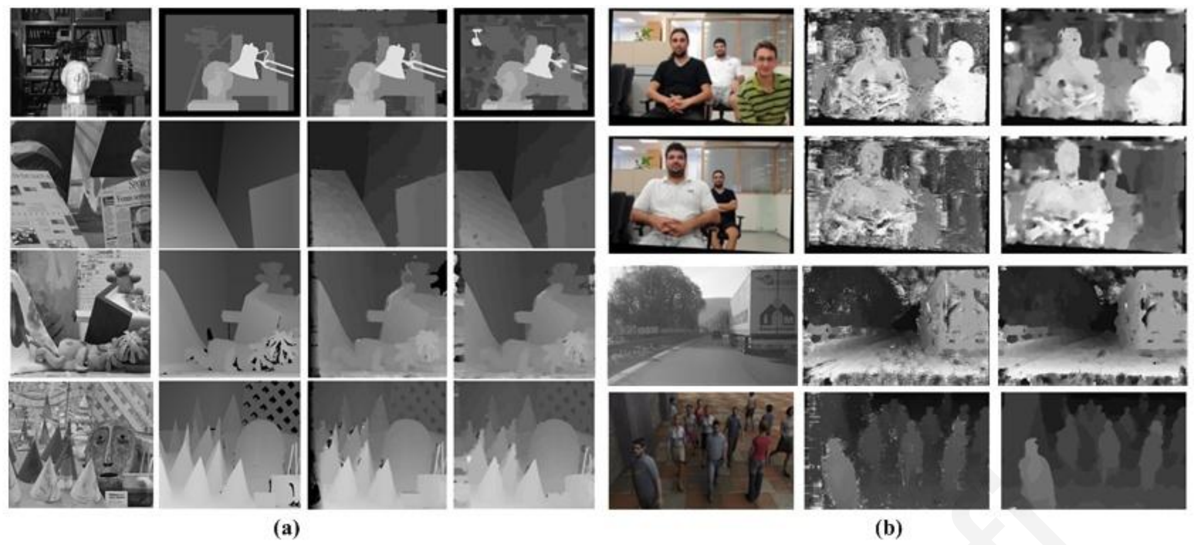


**Figure 4.8: Evaluation results with synthetic and real-world images. (a) Evaluation results of the proposed system architecture and the original algorithm [Tombari et al. 2007] using Middlebury stereo pairs. From left to right: Reference image, Ground truth, Disparity maps generated by the original algorithm, and Disparity map yielded by the proposed system. (b) Evaluation results of real-world and synthetic scenes before and after applying post-processing steps. Real-world scenes captured in our laboratory (rows 1-2), a real-world scene with a moving vehicle (row 3), and a synthetic scene of pedestrians (row 4).**

Compared to the original segment-based ADSW algorithm presented in [**63**], the proposed architecture performs reasonably well in terms of disparity estimation accuracy, especially at regions with enough details (non-occluded regions). In particular, the distance in quality between the original software implementation and the proposed architecture is only 3.35% on average (1.42% at *nonocc* regions, 3% at *all* regions, and 5.66% at *disc* regions). As shown in [**61**], the segment-based ADSW algorithm implemented on Intel Core Duo 2.14GHz CPU with 2GB RAM requires ~33 minutes to process a 450x375 stereo pair. On the other hand, the proposed hardware architecture along with the hardware-directed optimization techniques has significant speedup improvements (more than 10,000x), which justify the small quality reduction, and render our architecture suitable for embedded computing systems requiring high quality disparity maps in real time.

Furthermore, we compare the quality of the generated disparity maps with some related works. Note that some implementations listed in Table 4.4 are omitted from Table 4.5 and Table 4.6, because either they do not present quality results, or they have adopted a different metric, and thus cannot be directly compared to this work. As shown in Table 4.6, the proposed hardware architecture offers better quality (lower percentage of bad pixels) when compared to the DSP implementation in [**99**] and the dedicated hardware implementations in [**105**], [**107**], [**109**], [**108**], which adopt fixed-support stereo correspondence algorithms. It also

**Table 4.5: Comparison of the percentage of bad matching pixels between different implementations.**

| | Tsukuba | | | Venus | | | Teddy | | | Cones | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **nonocc (%)** | **all (%)** | **disc (%)** | **nonocc (%)** | **all (%)** | **disc (%)** | **nonocc (%)** | **all (%)** | **disc (%)** | **nonocc (%)** | **all (%)** | **disc (%)** |
| **Baha [2011]** | 3.94 | 5.84 | n.a | 5.03 | 6.36 | n.a | 13.68 | 21.49 | n.a | 6.59 | 15.66 | n.a |
| **Tombari [2007]** | 1.25 | 1.62 | 6.68 | 0.25 | 0.64 | 2.59 | 8.43 | 14.2 | 18.2 | 3.77 | 9.87 | 9.77 |
| **Gehrig [2010]** | n.a. | 3.38 | n.a. | n.a. | 3.29 | n.a. | n.a. | 16.54 | n.a. | n.a. | 13.04 | n.a. |
| **Chang [2007]** | 21.5 | 21.7 | 48.7 | 16.5 | 17.8 | 29.9 | 26.3 | 33.6 | 35,1 | 24.2 | 32.4 | 31.0 |
| **Humenberger [2010]** | 5.08 | 6.25 | 19.20 | 1.58 | 2.42 | 14.20 | 7.96 | 13.80 | 20.30 | 4.10 | 9.54 | 12.20 |
| **Yang [2006]** | 1.49 | 3.40 | n.a | 0.77 | 1.90 | n.a | 8.72 | 13.2 | n.a | 4.61 | 11.6 | n.a |
| **Wang [2006]** | 2.05 | 4.22 | 10.60 | 1.92 | 2.98 | 20.30 | 7.23 | 14.40 | 17.60 | 6.41 | 13.70 | 16.50 |
| **Xiang [2012]** | 1.67 | 3.82 | 8.44 | 0.63 | 1.67 | 8.67 | 7.76 | 14.7 | 17.8 | 4.29 | 12.4 | 11.4 |
| **Georgoulas [2009]** | n.a | 13.55 | n.a | n.a | 12.60 | n.a | n.a | n.a | n.a | n.a | 12.60 | n.a |
| **Darabiha [2006]** | 19.59 | n.a | 37.62 | 10.51 | n.a | 31.52 | n.a | n.a | n.a | n.a | n.a | n.a |
| **Jin [2010]** | 9.79 | 11.56 | 20.29 | 3.59 | 5.27 | 36.82 | 12.50 | 21.50 | 30.57 | 7.34 | 17.58 | 21.01 |
| **Ambrosch [2010]** | 5.81 | 7.14 | 22.60 | 2.61 | 3.33 | 25.30 | 9.79 | 15.50 | 25.70 | 5.08 | 11.50 | 15.0 |
| **Chang [2010]** | n.a | 2.80 | n.a | n.a | 0.64 | n.a | n.a | 13.70 | n.a | n.a | 10.10 | n.a |
| **Ding [2011]** | 3.73 | 5.65 | 10.30 | 1.59 | 3.46 | 10.40 | 13.50 | 20.60 | 20.90 | 10.80 | 18.20 | 19.0 |
| **Gehrig [2009]** | n.a. | 5.86 | n.a. | n.a. | 3.85 | n.a. | n.a. | 13.28 | n.a. | n.a. | 9.54 | n.a. |
| **Banz [2011]** | 6.8 | n.a. | n.a. | 4.1 | n.a. | n.a. | 13.3 | n.a. | n.a. | 9.5 | n.a. | n.a. |
| **Hirschmuller [2012]** | 2.98 | n.a. | n.a. | 1.49 | n.a. | n.a. | 6.36 | n.a. | n.a. | 3.57 | n.a. | n.a. |
| **Ttofis [2012]** | 4.48 | 6.04 | 12.7 | 6.01 | 7.47 | 18.2 | 21.5 | 28.1 | 28.8 | 17.1 | 25.9 | 25.8 |
| **Proposed Work** | 3.99 | 5.00 | 16.5 | 0.84 | 1.94 | 6.85 | 9.52 | 17.3 | 22.0 | 5.03 | 14.1 | 14.5 |

outperforms the software implementation of [**136**], [**137**], which implements a variable support method, while compares favorably with the DSP implementation based on optimized Census transform in [**65**]. Moreover, it compares favorably with the real-time implementation of the SGM algorithm on the CPU [**90**] (9.59% vs. 9.06% in non-occlusion regions). Furthermore, the proposed architecture achieves comparable quality to the GPU implementation in [**93**] that combines the ADSW approach with Dynamic programming, and a small quality reduction (~2%) compared to the GPU implementation in [**94**] that implements a global optimum disparity estimation algorithm based on improved belief propagation. The GPU-based system in [**94**] indicates that complicated disparity estimation algorithms implemented on GPUs can achieve very good quality results; however, the high levels of quality come at the expense of limited processing performance, as [**94**] obtained ~10 fps for small-sized images (384x288).

When compared to hardware-based methods of ADSW algorithms, the proposed architecture outperforms the implementation in [**115**]. It is worth noting that this implementation presents a quality reduction of 4.84% compared to the original algorithm adopted (as presented in [**115**]). This demonstrates the effectiveness of the specific hardware-directed optimization techniques adopted in the proposed implementation (quality reduction relative to original algorithm = 3.35%) in keeping a balance between hardware complexity and accuracy. The other ADSW implementation [**114**] outperforms the proposed implementation when considering *all* regions. However, the implementation in [**114**] does not provide quantitative results for the error rate at depth discontinuity regions. These regions though, are significantly important when implementing ADSW algorithms, as the idea of the adaptive support weight approach is primarily motivated by the need to accurately detect depth borders (where depth discontinuities occur). As such, the effectiveness of the implementation in [**114**] cannot be directly compared to the proposed implementation. In addition, the implementation in [**114**] focuses only on synthetic image data. As evidence in [**29**] however, methods that work well on synthetic scenes might not work well on real-world scenes. In this work, we also provide evaluation results for real-world scenes in Figure 4.8 (b), evidencing the effectiveness of the proposed implementation.

Finally, it must be acknowledged that the dedicated hardware implementations of SGM-based methods in [**110**], [**113**] achieve slightly better quality results compared to the proposed implementation when considering specific kind of regions (*all* and *nonocc* regions, respectively). However, they are both slower in terms of processing performance. On the other hand, the SGM-based implementation in [**112**] has similar processing performance to our implementation, but obtains a lower average percentage of bad matching pixels at *nonocc* regions (quality results for the other kind of regions are not provided). It must be noted though, that while the system in [**112**] includes a simple post-processing pipeline (LR check and 3x3 median filtering), the modular nature of the architecture can allow the integration of more sophisticated post-processing steps (e.g. hole interpolation) to obtain higher accuracy.

#### 4.5.4.3 *Discussion*

In order to highlight the implementations that better tradeoff between accuracy and processing speed, we followed a similar ranking scheme to the one proposed in [**61**] and list in Table 4.6 an average performance ranking obtained by averaging the accuracy ranking (based on the percentage of bad pixels in the leftmost columns in the table) and the processing speed ranking (based on the MDE/s values in Table III). In particular, the different implementations are ranked in terms of processing speed based on the MDE/s values from Table 4.4, and also in terms of accuracy based on the average percentage of bad matching pixels from columns 1-4 of Table 4.6. This ranking is listed in columns 5-9 of Table 4.6, and is then used to derive the average ranking (columns 10-13) by averaging the speed ranking (column 5 in Table 4.6) with the overall  accuracy ranking (column 6 in Table 4.6) and the accuracy ranking at *nonocc*, *all*

**Table 4.6: Average performance ranking.**

| | Percentage of Bad Pixels[1] | | | | Ranking[2] | | | | | Average Ranking[3] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **avg** | **nonocc** | **all** | **disc** | **MDE/s** | **avg** | **nonocc** | **all** | **disc** | **avg** | **nonocc** | **all** | **disc** |
| **Baha [2011]** | n.a | 7.31 | 12.34 | n.a | 18 | n.a | 9 | 12 | n.a | n.a | 13.5 | 15.0 | n.a |
| **Tombari [2007]** | 6.44 | 3.43 | 6.58 | 9.31 | 19 | 1 | 1 | 1 | 1 | 10.0 | 10.0 | 10.0 | 10.0 |
| **Gehrig [2010]** | n.a. | n.a. | 9.06 | n.a. | 12 | n.a. | n.a. | 8 | n.a. | n.a. | n.a. | 10.0 | n.a. |
| **Chang [2007]** | 28.23 | 22.13 | 26.38 | 36.18 | 11 | 11 | 15 | 16 | 11 | 11.0 | 13.0 | 13.5 | 11.0 |
| **Humenberger [2010]** | 9.72 | 4.68 | 8.00 | 16.48 | 15 | 4 | 6 | 4 | 6 | 9.5 | 10.5 | 9.5 | 10.5 |
| **Yang [2006]** | n.a | 3.90 | 7.53 | n.a | 16 | n.a | 4 | 3 | n.a | n.a | 10.0 | 9.5 | n.a |
| **Wang [2006]** | 9.83 | 4.40 | 8.83 | 16.25 | 14 | 6 | 5 | 7 | 5 | 10.0 | 9.5 | 10.5 | 9.5 |
| **Xiang [2012]** | 7.77 | 3.59 | 8.15 | 11.58 | 17 | 2 | 2 | 6 | 2 | 9.5 | 9.5 | 11.5 | 9.5 |
| **Georgoulas [2009]** | n.a | n.a | 12.92 | n.a | 1 | n.a | n.a | 13 | n.a | n.a | n.a | 7.0 | n.a |
| **Darabiha [2006]** | n.a | 15.05 | n.a | 34.57 | 13 | n.a | 14 | n.a | 10 | n.a | 13.5 | n.a | 11.5 |
| **Jin [2010]** | 16.49 | 8.31 | 13.98 | 27.17 | 3 | 9 | 11 | 14 | 9 | 6.0 | 7.0 | 8.5 | 6.0 |
| **Ambrosch [2010]** | 12.45 | 5.82 | 9.37 | 22.15 | 5 | 8 | 8 | 9 | 8 | 6.5 | 6.5 | 7.0 | 6.5 |
| **Chang [2010]** | n.a | n.a | 6.81 | n.a | 9 | n.a | n.a | 2 | n.a | n.a | n.a | 5.5 | n.a |
| **Ding [2011]** | 11.51 | 7.41 | 11.98 | 15.15 | 6 | 7 | 10 | 11 | 4 | 6.5 | 8.0 | 8.5 | 5.0 |
| **Gehrig [2009]** | n.a. | n.a. | 8.13 | n.a. | 10 | n.a. | n.a. | 5 | n.a. | n.a. | n.a. | 7.5 | n.a. |
| **Banz [2011]** | n.a. | 8.43 | n.a. | n.a. | 4 | n.a. | 12 | n.a. | n.a. | n.a. | 8.0 | n.a. | n.a. |
| **Hirschmuller [2012]** | 8.84 | 3.6 | n.a. | n.a. | 8 | 3 | 3 | n.a. | n.a. | 5.5 | 5.5 | n.a. | n.a. |
| **Ttofis [2012]** | 16.84 | 12.27 | 16.88 | 21.38 | 7 | 10 | 13 | 15 | 7 | 8.5 | 10.0 | 11.0 | 7.0 |
| **Proposed FPGA System** | 9.79 | 4.85 | 9.59 | 14.96 | 4 | 5 | 7 | 10 | 3 | 4.5 | 5.5 | 7.0 | 3.5 |
| **Proposed ASIC System[4]** | | | | | 2 | | | | | 3.5 | 4.5 | 6.0 | 2.5 |

[1] Average percentage of bad pixels computed over all twelve columns (*avg*) and over columns *nonocc*, *all* and *disc* in Table 4.5, respectively.
[2] Ranking of different implementations based on the MDE/s values from Table 4.4, and on the average percentage of bad matching pixels from columns 1-4 of Table 4.5.
[3] Ranking obtained by averaging the speed ranking (column 5 in Table 4.6) with the overall accuracy ranking (column 6 in Table 4.6) and the accuracy ranking at *nonocc*, *all* and *disc* regions (columns 7-9 in Table 4.6), respectively.
[4] Fully parallel system that processes 640x480 images and supports a disparity range of 64 (Configuration 2 in Table 4.3).

and *disc* regions (columns 7-9 in Table 4.6), respectively. One particular observation is that our architecture is ranked third in the depth discontinuity regions (*disc*) after the original algorithm proposed in [**63**] and the GPU-based implementation of a global hierarchical BP algorithm in [**94**], outperforming all dedicated-hardware implementations. Overall, by looking at the rightmost columns of the last row in Table 4.6, it is observed that our architecture can be regarded as an effective accuracy-speed tradeoff, as its average ranking values (and especially the values associated with the ASIC implementation) are the smallest when considering the *avg*, *nonocc* and *disc* columns, and among the smallest when considering the *all* column. As such, the proposed system provides high disparity map estimation speed, while it is ranked among the top methods in terms of accuracy. Therefore, the proposed hardware architecture indicates high potential for high-level vision applications in emerging embedded computing systems such as intelligent robotic, surveillance, automotive and navigation systems, which require good performance in both accuracy and speed. Moreover, given that our system obtains improved accuracy at depth discontinuity points, it might be more suitable for tasks such as recognition or robotic grasping, where it is particularly important to obtain good accuracy near such points.

## 4.6 Concluding Remarks

This chapter investigated the use of adaptive support weights in embedded stereo matching architectures. Initially, a discussion on how weighting the pixels in the support window can increase the robustness of the matching process, improving the quality in regions with low texture and at depth discontinuities, was provided. The chapter also discussed the complications of adaptive weights for hardware implementations and showcased how such algorithms can be adopted for hardware-friendly designs. It is worth mentioning that the edge-directed approach presented in section Chapter 3 can also be utilized in ADSW hardware implementations as well. Although it is not shown in this chapter, the reader is referred to Appendix B, which describes an ADSW stereo matching architectures that also utilizes edge information.

The proposed ADSW-based stereo matching architecture was implemented on an FPGA platform, evaluated with real-time data acquisition using a custom-built stereo camera setup, and compared with related dedicated hardware implementations in terms of accuracy, processing speed and hardware resource utilization. Experimental results were very encouraging in that the complete system is able to provide an effective accuracy/speed tradeoff. However, this was achieved at the expense of relatively high resource usage, something that motivated us to introduce the hardware architectures presented in the following chapter.

# Chapter 5

# High-Quality Real-Time Hardware Stereo Matching Based on Guided Image Filtering

*SEVERAL emerging embedded vision applications require high-quality depth computation and real-time frame-rate. By implementing a simplified version of an adaptive support weight (ADSW) algorithm, the stereo matching hardware architecture presented in the previous chapter provides a good balance between accuracy and speed; however at the expense of high hardware demands and noticeable quality degradation compared to the original ADSW algorithm. This chapter investigates possibility of designing hardware-based stereo matching architectures able to provide high accuracy and concurrently high performance for embedded vision devices, which are associated with limited hardware and power budget. The proposed architectures integrate a compact and efficient design of the recently proposed guided image filter; an edge-preserving filter that reduces the hardware complexity of the implemented stereo algorithm, while at the same time maintains high-quality results. The guided filter design is utilized in different ways in the stereo matching processing pipeline, illustrating its potential in reducing the complexity of the ADSW aggregation process, but also its efficiency in enabling a powerful disparity refinement unit, which can improve the matching accuracy considerably, even if the cost aggregation is based on simple, fixed support strategies. Prototypes of the architectures have been implemented on a Kintex-7 FPGA board, achieving real-time processing speed (60 fps) even for high definition 720p resolution video. Moreover, the proposed stereo matching designs deliver leading accuracy when compared to state-of-the-art hardware implementations.*

## 5.1 Introduction - Motivation

High-quality real-time stereo matching has the potential to enable various applications in emerging embedded vision systems including fully autonomous robotics, three-dimensional video surveillance and security. However, such systems usually require real-time processing capability under limited resources (e.g. memory and power), something that does not permit the use of most sophisticated stereo matching approaches. In recent years, a fair amount of work has been carried out on real-time hardware implementations of local stereo matching algorithms (e.g. [**108**]); a thorough review is presented in [**85**]. The majority of these implementations have adopted standard block-based aggregation with fixed support windows (in size and/or shape) [**85**]. These algorithms can achieve very high frame rates when implemented in hardware [**85**], but they lead to low matching accuracy [**62**]. High matching accuracy though is of foremost importance in many of today's embedded vision applications. As such, a few attempts have been made recently directed towards improving the matching accuracy, either by combining different stereo algorithms together, or by implementing modified versions of Semi Global Matching (SGM) [**138**], [**139**] and local Adaptive Support Weight (ADSW) algorithms [**114**], [**115**], [**68**].

The hardware implementation in [**109**] performs a modified version of the Census transform in both the intensity and gradient images, in combination with the SAD correlation metric. An FPGA implementation of a stereo algorithm based on the neural network and Disparity Space Image (DSI) data structure is introduced in [**137**]. The real-time FPGA-based stereo matching design presented in [**140**] combines the mini-Census transform and the Cross-based cost aggregation. SGM-based stereo matching systems have been introduced in [**138**], [**139**] and implemented on FPGAs and a hybrid FPGA/RISC architecture, respectively. The technical details/parameters of the different implementations are summarized in Table 5.4.

The works that are closely related to ours in terms of the matching algorithm are the works in [**114**], [**115**], [**68**], which implement ADSW-based algorithms. The segment-based ADSW architecture presented in Chapter 4 also belongs to this latter category. The work in [**114**] proposed the VLSI design of a hardware-friendly ADSW algorithm that adopted the mini-Census transform to improve the accuracy and robustness to radiometric distortion. The work in [**115**] proposed the implementation of a complete stereo vision system, which incorporates an ADSW algorithm and also integrates pre- and post- processing units. Finally, a hardware-oriented stereo matching system based on the adaptive Census transform is presented in [**68**]. The aforementioned high-quality ADSW-based systems follow a similar algorithm-to-hardware mapping methodology. That is, a complex, but accurate, algorithm is adapted for dedicated hardware implementation through a series of algorithmic modifications/approximations. In most cases, however, these implementations scarify part of the accuracy; quality reduction compared to the original implementation of the ADSW approach in [**62**] is ~ 4-5%. In addition, their high memory and hardware demands limit their scalability to higher resolution images.

Recently, the idea to utilize the Guided Image Filter (GIF) [**141**] in local ADSW stereo matching algorithms has been proposed to reduce the complexity of the cost aggregation step. The GIF has also been utilized to perform weighted median filtering during the disparity refinement step. Such software implementations have yielded promising results [**142**], [**143**]. Motivated by the results of the software implementation presented in [**142**], [**143**], this work focuses on designing fully pipelined, parallel and scalable stereo matching hardware architectures that integrate the recently proposed GIF. The thesis aims to present a new and efficient hardware design of the GIF (that can be potentially adopted in other uses of the filter), and also to explore and concurrently discuss the hardware design parameters and optimizations involved in integrating the GIF hardware architecture in the cost aggregation and disparity refinement steps of hardware-based local stereo matching systems. The latter is expected to reduce the overall hardware complexity of cost aggregation, which in turn will allow real-time stereo matching of high definition images (HD), as well as improvements of the overall matching accuracy, thanks to the edge-preserving property of the GIF. Due to this type of filter, and its optimized hardware design presented in this work, the proposed architecture(s) push further the accuracy limits of hardware-based stereo matching systems, while they also achieve real-time frame-rates for HD images.

Additionally, the thesis relies on FPGA implementation and emulation to perform a detailed benefit-cost analysis (in terms of matching quality vs. resource usage given a fixed throughput constraint) of different system configurations (e.g. the GIF is integrated either in cost aggregation/disparity refinement or in both steps). The configuration found to better exploit the benefits of the GIF is developed as part of a complete stereo vision system on a Kintex-7 FPGA board. The prototype supports 720p@60Hz HD video sequences (captured from a custom-built stereoscopic camera system) and various disparity ranges. Finally, the selected GIF-based Stereo Matcher is evaluated qualitatively and quantitatively, based on Middlebury benchmark image sets [**25**], and is also compared with existing state-of-the-art stereo matching hardware systems.

The rest of this Chapter is organized as follows: Section 5.2 provides background information on the GIF and its use in stereo matching. Section 5.3 presents a compact and efficient hardware design of the GIF. Section 5.4 discusses how the GIF design can be integrated into stereo matching hardware architectures. Section 5.5 shows results and comparison with related work. Finally, Section 0 concludes the chapter.

## 5.2 The Guided Image Filter and its Use in Stereo Matching

This section provides background information related to the guided image filter. A brief definition of the filter is first introduced along with the necessary mathematical formulation. The guide filter is quite useful in several applications in image processing and computer vision [**141**]. Here, we will focus on the application of this filter in the stereo matching process, and in particular, its integration in the cost aggregation and disparity refinement steps, along with associated benefits for dedicated hardware implementations.

### 5.2.1 The Guided Image Filter (GIF)

The Guide Image Filter (GIF) generally uses a *guidance* image *I* to filter a *guided* image *p*; though *I* and *p* can be identical. A general equation expressing the output of a linear filter at each pixel *i*, is defined in (5.1) as a weighted average of the filter kernel $W_{i,j}$ and the guided image $p$. The filter kernel $W_{i,j}$ depends entirely on the guidance image; it is a function of *I*, independent of *p*. The filtering output $q$ is linear with respect to the guided image *p*. The following paragraphs will focus on the definition of the guided image filter. Equation (5.1) will be revisited at a later stage to define the filter kernel.

$$q_i = \sum_j W_{i,j}(I)p_j \quad i,j\text{: } pixel\ indices \tag{5.1}$$

The working principle of the GIF lies on the assumption that the filtering output $q$ is generated based on a local linear model between *I* and *p*. The filtering output *q* can be defined as a linear transform of *I* in a window $\omega_k$ centered at the pixel *k* by finding a solution to (5.2), where $a_k$ and $b_k$ are some linear coefficients assumed to be constant in $\omega_k$. By definition $\omega_k$ is a square window with integer radius *r*, and a side length $2r + 1$ ($|\omega_k|$ is the number of pixels in $\omega_k$). The linear model describing the GIF in (5.2) makes the filter an edge-preserving filter, because $\nabla q = a_k I$ (since $a_k$ and $b_k$ are constants). In other words, the filtered output image *q* has an edge only if the guidance image *I* has also an edge.

$$\exists a_k, b_{k,} \forall i \epsilon \omega_k \qquad q_i = a_k I + b_k \tag{5.2}$$

A solution to (5.2) can be determined by finding the linear coefficients that minimize the difference between q and the filtering input p. This is achieved by modeling the output q as the input p, subtracting unwanted noise / texture, as given in (5.3). The coefficients are then determined by minimizing the cost function $E\left(a_k, b_{k,}\right)$ in the window $\omega_k$ as in (5.4), where $\epsilon$ is a regulization parameter preventing $a_k$ from being too large. This minimization function is solved by following the linear ridge regression model [**141**], and its solution is given in (5.5) & (5.6). The terms $\mu_k$ and $\sigma_k^2$ represent the mean and variance of I in $\omega_k$.

$$q_i = p_i - n_i \tag{5.3}$$

$$\min_{a_k, b_{k,}} \left( E\left(a_k, b_{k,}\right) \coloneqq \sum_{i \epsilon \omega_k} \left( (a_k I_i + b_k - p_i)^2 + \epsilon a_k^2 \right) \right) \tag{5.4}$$

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \epsilon \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \varepsilon} = \frac{cov\,(I,p)}{var\,(I) + \varepsilon} \tag{5.5}$$

$$b_k = \bar{p}_k - a_k \mu_k = mean(p) - a \cdot mean(I) \tag{5.6}$$

Having computed the linear coefficients $a_k$ and $b_k$, the filtering output $q_i$ can be computed by replacing them in (5.2). However, in order to reach the final definition of the GIF, it is necessary to account for the fact that any pixel $i$ is coved by several overlapping windows $\omega_k$, which leads to different values of $q_i$, when this is computed at different windows (due to many values of the coefficients $a_k$ and $b_k$). This problem is solved using the averaging strategy of overlapping windows, which is popular in image denoising [**15**]. Based on this strategy, the filtering output is redefined as in (5.7), where $\bar{a}_k$ and $\bar{b}_k$ are the mean values of $a_k$ and $b_k$ in the window $\omega_k$, respectively ($i\epsilon\omega_k$ and $k\epsilon\omega_i$ are equivalent due to the symmetry of the box window).

$$q_i = \frac{1}{|\omega|} \sum_{k, i \epsilon \omega_k} q_i \;= \frac{1}{|\omega|} \sum_{k, i \epsilon \omega_k} (a_k I + b_k) = \bar{a}_k I + \bar{b}_k \tag{5.7}$$

It must be noted that $\nabla q$ is no longer scaling with respect to $\nabla I$, as the coefficients $\bar{a}_k$ and $\bar{b}_k$ vary spatially. However, since these coefficients are the output of mean filters, their gradients are expected to be much smaller near strong edges than the gradient of $I$. Hence, we can still assume that $\nabla q \cong \bar{a} I$ , which means than the edge-preserving property is mostly preserved (i.e. sudden intensity changes in $I$ most likely will be preserved in $q$).

The overall guided filtering process is illustrated in Figure 5.1. Besides the edge-preserving property, another inherent advantage of this type of filter is that the operations involved in computing the filtering output are linear, and can be decomposed into a series of

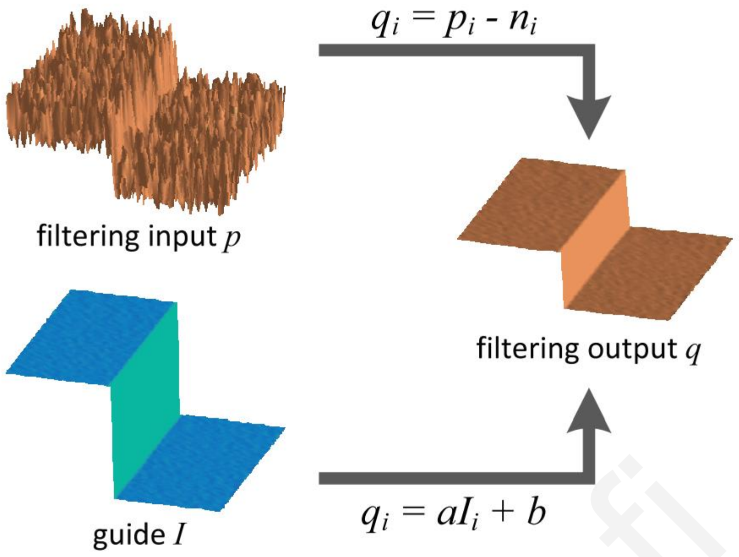


**Figure 5.1: Illustration of the guided image filtering process [141].**

mean filters with windows radius $r$. This is a very important property that leads to a fast implementation of the GIF, because mean filtering (box filtering) can be efficiently performed using integral images or moving sums in O(1) time (with respect to $r$). Algorithm 5.1 provides the pseudocode of the GIF algorithm, which mainly consists of four steps:

1. Computing the images corresponding to the mean values of the guidance image $I$ and the guided image $p$.
2. Computing the images corresponding to the variance of image $I$ and the covariance of images $I$ and $p$.
3. Computing the images corresponding to $a_k$ and $b_k$.
4. Computing the images corresponding to $\bar{a}_k$ and $\bar{b}_k$.
5. Computing $q$ based on (5.7).

The GIF presented above is completely defined by equations (5.5)-(5.7). It is worth noting the relationships between $I$, $p$ and $q$ in these equations, are in the weighted-average form of (5.1). In fact, all three equations can be rewritten as weight sums of the forms $a_k = \sum_j A_{kj}(I)p_j$, $b_k = \sum_j B_{kj}(I)p_j$ and $q_i = \sum_j W_{ij}(I)p_j$, respectively. It can be proved (the proof is provided in [**141**]) that the kernel weights can be explicitly expressed as in (5.8). This equation suggests that pixels $j$ in the neighborhood of $i$ have a stronger weight in the resulting value of $q_i$ if they share similar color with $i$.

**Algorithm 5.1: Guided Image Filter**

```
Input: guidance image $I$, guided image $p$
Output: filtering output $q$
Parameters: $r$, $\varepsilon$
1:  $mean_I = f_{mean}(I)$;  $mean_p = f_{mean}(p)$;
    $corr_I = f_{mean}(I.*I)$;  $corr_{Ip} = f_{mean}(I.*p)$;
2:  $var_I = corr_I - mean_I.*mean_I$;
    $cov_{Ip} = corr_{Ip} - mean_I.*mean_p$;
3:  $a = cov_{Ip}/(var_I + \varepsilon)$;  $b = mean_p - a.*mean_I$;
4:  $mean_a = f_{mean}(a)$;  $mean_b = f_{mean}(b)$;
5:  $q = mean_a.*I + mean_b$;
```

$$W_{ij}(I) = \frac{1}{|\omega|^2} \sum_{k:(i,j)\epsilon\omega_k} \left(1 + \frac{(I_i - \mu_k)(I_j - \mu_k)}{\sigma_k^2 + \varepsilon}\right) \tag{5.8}$$

## 5.2.2 Stereo Matching using Guided Image Filtering

The edge-aware nature of the Guided Image Filter (GIF) along with the fact that it can be implemented using a cascade of mean filters (as suggested by its pseudocode in Algorithm 5.1), has recently led to its integration in stereo matching algorithms, as a means to improve the matching quality in a computationally efficient manner. In particular, the work in [**142**] employed the GIF to reduce the complexity of the cost aggregation step in ADSW methods, while the work in [**143**] has exploited the filter to perform weighted median filtering during the disparity refinement step. This section discusses how the GIF can be integrated in the four-step stereo matching processing pipeline. In subsequent sections, we introduce a new and efficient hardware design of the filter, and also explore the hardware design parameters and optimizations involved in integrating the filter in stereo matching hardware architectures.

The GIF is an edge-aware smoothing filter, and as such, in order to better understand how it is possible to integrate it in the stereo matching process, we will revisit the four-step stereo matching processing pipeline and look at it in a different way. Until now, the cost aggregation step has been considered as summation of pixel dissimilarities inside support windows. However, there is a completely different way for looking at this step, which lies on the concept of the Stereo Cost Volume (SCV) or Disparity Space Image (DSI). The SVC is basically a 3D data structure constructed by computing the pixel dissimilarities between the reference image of the stereo pair and a number of shifted versions of the target image, at the different disparity levels. According to this structure, the cost aggregation step can be interpreted as a smoothing of each 2D slice of the SCV. From this perspective, the different aggregation methods differ mainly in the kernel used to accomplish this smoothing. Aggregation with a fixed correlation window is equivalent to applying a typical mean filter, which is fast, but, fails at preserving object boundaries. On the contrary, ADSW need to apply a weighted smoothing filter at each slice of the SVC. Fortunately, the GIF can be utilized to smooth the SVC in ADSW methods, giving an edge-preserving effect with high quality, while at the same time, it can be implemented using the mean filter as its basic building block.

Figure 5.2 gives an outline of the stereo matching processing pipeline that integrates the concept of the SVC discussed in the previous paragraph, while the pseudocode of the algorithm is provided in Algorithm 5.2. As shown, the algorithm consists of four major steps:

**Cost Volume Construction.** This step calculates a matching cost for each pixel $(i, j)$ at all possible disparities. The output is a three-dimensional structure (the SVC mentioned above) consisting of D cost images. Each cost can be computed based on the different correlation metrics listed in Table 2.1 (Section 2.5.1). In this work, we assume that each cost is computed as the truncated absolute difference of colors and gradients, a metric that exhibits good robustness to illumination changes [**142**]. We also adopt this metric for comparison purposes between the GIF-based stereo matching architectures developed in this work and existing software-based implementations [**142**], [**143**] that also adopt the GIF. The overall cost function C(p,d) is computed with (5.9)-(5.11), where $a$ is used to balance the influence of the color and gradients terms, and $T_c$ and $T_g$ are truncation thresholds.

$$M(i,j,d) = \sum_{i=1}^{3} \left| I_{left}^{i}(p) - I_{right}^{i}(p-d) \right| \tag{5.9}$$

$$G(i,j,d) = \left| \nabla_x\left(I_{left}(p)\right) - \nabla_x\left(I_{right}(p-d)\right) \right| \tag{5.10}$$

$$C(i,j,d) = a \cdot in\left(T_c, M(i,j,d)\right) + (1-a) \cdot min\left(T_g, G(i,j,d)\right) \tag{5.11}$$

**Cost Volume Filtering.** This step utilizes the GIF to smooth each slice of the SCV. Due to its edge-preserving property, the GIF leads to good accuracy at depth discontinuities. Typically, the filtered cost value at $q$ and disparity $d$ is a weighted average of the pixels in the same slice of the SCV, and is expressed as in (5.12).

$$q(i,j,d) = \sum W_{i,j}(I) C(i,j,d) \tag{5.12}$$

The GIF generally uses the reference image (grayscale version) of the stereo pair as the *guidance* image $I$ to filter a slice of the SVC, which in this case, is the *guided* image $f$. The filter weights are defined as in (5.8), where $\mu_k$ and $\sigma_k$ are the mean and the variance of $I$ in a

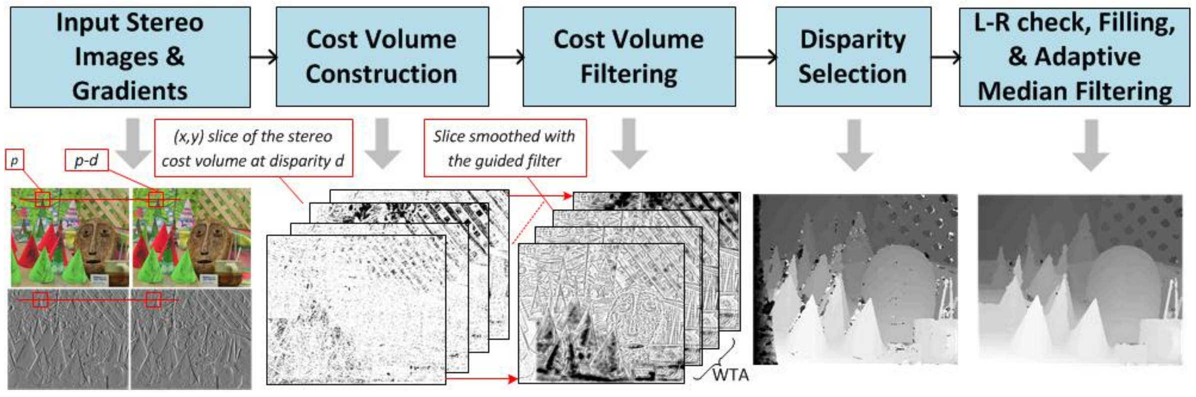

Input Stereo Images & Gradients
Cost Volume Construction
Cost Volume Filtering
Disparity Selection
L-R check, Filling, & Adaptive Median Filtering
p
p-d
(x,y) slice of the stereo cost volume at disparity d
Slice smoothed with the guided filter
WTA

**Figure 5.2: Major Steps of the GIF-based Stereo Matching Algorithm.**

squared window $\omega_k$ with dimensions $r \times r$, centered at pixel $k$. $|\omega|$ is the number of pixels in the window and $\varepsilon$ is a smoothness parameter. As already discussed, an inherent advantage of the GIF is that the weights can be computed with some linear operations (see [**141**]), which can be decomposed into a series of mean filters with windows radius $r$.

**Disparity Selection.** Once the SCV slices are filtered, the best disparity for pixel $p$ is chosen through a simple Winner-Takes-All (WTA) minimization approach using (5.13).

$$d_p = \underset{d \in D}{\operatorname{argmin}}\, q(i,j,d) \tag{5.13}$$

**Disparity Refinement.** The disparity map generated in the previous step is generally a noisy disparity map, hence, the disparity refinement step aims to reduce the noise and improve the overall accuracy. A left/right consistency check (L-R check) is performed. Therefore, the disparity map, $D_R$, using the right image as reference is also computed. The L-R check marks disparities as invalid if the disparity $D_L(i,j)$ and its corresponding disparity of $D_R(i,j)$ differ by more than 1 pixel. Invalid pixels are then filled with the minimum disparity between their closest consistent pixels in the left and right direction. Weighted and typical median filtering are applied next to smooth the filled regions and remove spikes. The following section discusses how it is possible to perform adaptive median filtering using the GIF, thus improving further the effectiveness of the disparity refinement step in improving the overall matching quality. It is worth noting that other adaptive median filters can also be utilized. For example, the segmentation-based adaptive median filter presented in Chapter 4.4.3, is also a good candidate. In this work, we developed hardware designs of both filters, and provide tradeoffs in terms of accuracy when integrating either or both filters in the proposed GIF-based stereo matching architecture(s).

**Algorithm 5.2: Stereo Matching based on Guided Image Filtering.**

```
Input: Color Stereo Images $I_r$ and $I_t$
Output: Disparity map $D_L(.)$ using the left stereo image as reference
Parameters: Disparity range, radius of match window $r$
1.  Compute grayscale images $I_{r\_gray}$ and $I_{t\_gray}$
2.  Compute the mean and variance images (terms $\mu_k$ and $\sigma_k$ in (5.5)) of the grayscale image $I_{r\_gray}$
3.  $Cost(\cdot) \leftarrow \infty$
4.  foreach $d \in [d_m, d_M]$ do
5.      Building slice d of the SVC:
6.      foreach pixel $i, j$ in $I_r$ do
7.          Compute the color term (5.9) and the gradient term (5.10)
8.          Compute the pixel cost (5.11) as a balanced combination of (5.9) and (5.10)
9.          Set the computed slice of the SVC in the guided image $p = C(i, j, d)$

10.     Stereo cost volume slice filtering using the guided filter:
11.     Compute the mean of $p$ (term $\bar{p}_k$ in (5.6)
12.     Compute the variance image of $I_r$ and the covariance of image $I_r$ and $p$ in (5.5)
13.     Compute the images of coefficients $a_k$ and $b_k$ (5.5), (5.6)
14.     Compute the images $\bar{a}_k$ and $\bar{b}_k$ in (5.7)
15.     Compute filtering output $q$ by (5.7)

16.     Disparity selection:
17.     foreach pixel $i, j$ in $I_r$ do
18.         if $Cost(i, j) > q(i, j)$ then
19.          $Cost(i, j) \leftarrow q(i, j)$
20.          $D_L(i, j) \leftarrow d$ (5.13)
21. Disparity refinement:
22. Left-right consistency check:
23.     Compute the disparity map $D_R(.)$ using the right stereo image as reference (lines 1-20)
24.     foreach pixel $i, j$ in $D_L(.)$ do
25.     if $|D_L(i, j) - D_R(i + D_L(i, j), j)| < LRC\_threshold$ → label disparity $i, j$ as invalid
26. Interpolate invalid disparities
27. Filter interpolated disparity map with weighted median filter
28. Filter with an un-weighted median filter
```

## 5.2.3 Weighted Median Filtering using the Guided Filter Weights

Besides its use in smoothing the stereo cost volume during cost aggregation, the guided image filter can also be utilized to develop an efficient weighted median filter that can be applied during disparity map refinement to remove outlier errors, while preserving edges/structures [**143**]. In the following, the theory behind the GIF-based weighted median filtering process is first introduced, and then the filter's properties that make it suitable for fast and high-quality disparity refinement are discussed.

### 5.2.3.1 *Median Filtering*

The (unweighted) median filter is a widely known filter used to perform some kind of noise reduction in digital images. Due to its ability in removing "outliers" like salt-and-pepper noise, it is often integrated as a pre-processing step to improve the results of later image processing operations. The median filter works by replacing the value of each pixel with the median of its neighbors. A straightforward way to compute the median involves sorting the pixel values in the window, and selecting the value in the middle of the sorted list. A more efficient way is to compute the median by constructing the histogram $h(p_c,.)$ of the pixels inside a square window around pixel $p_c$, using the equation in (5.14). In this equation, $N(p_c)$ refers to a local, usually square window around the central pixel, $p_c$, whose median is being computed, $I(p_c)$ is the pixel value, $i$ is the discrete bin index, and $\delta(\cdot)$ is the Kronecker delta function, which is 1 when the argument is 0, and is 0 otherwise. After constructing the histogram, it is quite easy to pick the median value.

$$h(p_c,i) = \sum_{p_j \epsilon N(p_c)} \delta(I(p_c) - i) \tag{5.14}$$

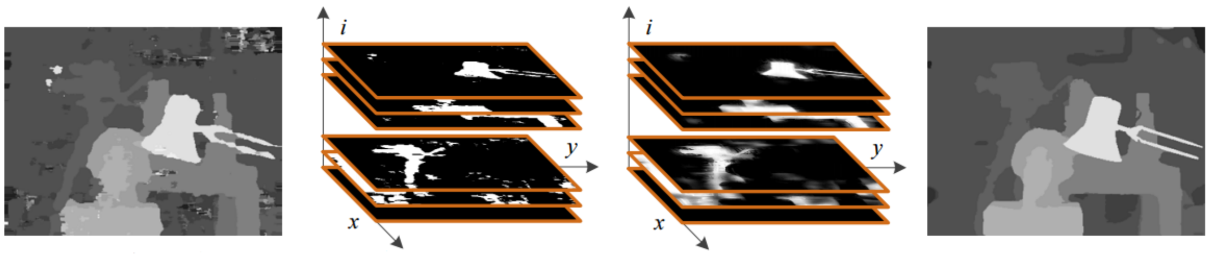
### 5.2.3.2 *Weighted Median Filtering*

The unweighted median filter gives equal importance to each neighbor pixel, and hence, it is possible to lead to morphological artifacts, for example removing thin structures or rounding sharp edges in the images. The weighted median filter aims to alleviate these problems by considering each neighbor pixel differently, through the use of weights assigned to each pixel. The weights are generated based on an image $J$ that can be different from the image to be filtered $I$. The median value is found by accumulating the weighted pixels in local histograms, as in (5.15).

$$h_w(p_c,i) = \sum_{p_j \epsilon N(p_c)} w(p_c,p_j)\delta(I(p_c) - i) \tag{5.15}$$

### 5.2.3.3 *GIF-based Weighted Median Filtering*

A naïve implementation of the weighted median filter is computationally demanding as it depends on the size of the local window and the discrete number of bins, which in practice is a big constant (256 for grayscale images), especially for high precision input images. An efficient constant time algorithm, which is independent of the window size, can be derived by considering the 3D signal $f(p_c,i)$ in (5.16), where $p_c$ is the central pixel with 2D spatial coordinates $(x,y)$ and $i$ is the bin index. With this signal, the computation of the histograms can essentially be performed using 2D filtering of signal $f$ in the spatial domain. This implies that the computation of the unweighted histogram in (5.14) can be simply calculated based on 2D box filtering (mean filtering) on $f$ as in (5.17), while the weighted histogram in (5.15) can be done by replacing the box filter with a weighted smoothing filter as in (5.18). The GIF



**(a) Noisy disparity map (b) The 3D signal $f(p_c, i)$ (c) Weighted histogram $h_w(p_c, i)$ (d) Refined Disparity Map**

**Figure 5.3: Weighted median filtering over a noisy disparity map based on the GIF [143].**

presented before is an excellent candidate for extracting the weights in (5.18). Its edge-aware property helps in maintain sharp edges during the median filtering process, but equally important, the GIF can have a fast implementation based on a series of box filters (as discussed in Section 5.2.1).

$$f(p_c, i) \triangleq \delta(I(p_c) - i) \tag{5.16}$$

$$h(p_c, i) = \sum_{p_j \epsilon N(p_c)} b(p_c, p_j) f(p_c, i) \tag{5.17}$$

$$h_w(p_c, i) = \sum_{p_j \epsilon N(p_c)} w(p_c, p_j) f(p_c, i) \tag{5.18}$$

### 5.2.3.4 *Integration with the stereo matching pipeline*

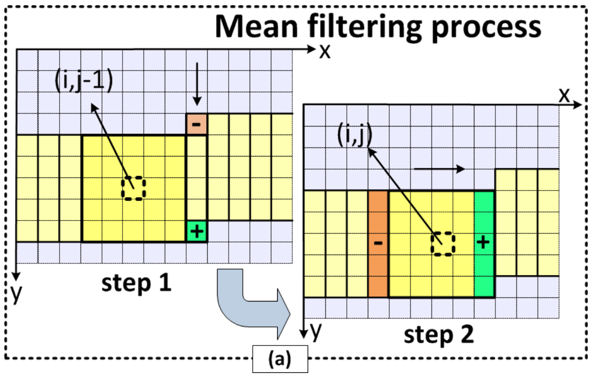
The GIF-based weighted median filter can be integrated in the stereo matching pipeline within the disparity refinement step to remove outliers from the disparity map, while maintaining edges/structures. This is possible by decomposing the disparity map extracted in the disparity computation step into $D$ binary images (D is the number of discrete disparity levels and also the number of bins). In other words, a binary image is generated for each fixed $i$, forming the 3D signal $f(p_c, i)$ in (5.17). Having computed this 3D signal, the weighted median filter can be done by applying a guided image filter on $f(p_c, i)$ for each fixed $i$. In this way, the final disparity result can be computed from the weighted histogram of $f(p_c, i)$. The weights of the guided filter in the refinement step are computed in the same way as they computed when applying the guided filter on each slice of the SVC in the cost aggregation step. In other words, they are computed based on the content of the reference image in the stereo pair. The overall process of weighted filtering using the GIF in disparity refinement is illustrated in Figure 5.3.

## 5.3 Hardware Implementation of the Guided Image Filter

Besides the high quality obtained by smoothing the SCV and/or refining the disparity map with the GIF, this type of filter can have an efficient dedicated hardware design, as the basic operation involved is the mean filter with windows of radius *r*. This section describes how the GIF is implemented in hardware efficiently in a way that its logic resources are independent of the kernel radius *r*.

The mean intensity of pixels over rectangular windows in the image can be implemented in a fast way using the integral image technique. However, this technique requires huge amount of memory, especially for high-resolution images. Therefore, we instead followed a variant of the approach in [**144**], to implement a custom mean filter design that consumes compact hardware resources. The main idea is to maintain a sum for each column in the image to be filtered. Each column sum accumulates 2r+1 pixels, while the window sum is computed by adding 2r+1 adjacent column sums. While filtering the image, the window sum is updated using the two-step approach illustrated in Figure 5.4 (a). When the window is moved to the right from one pixel to the next, the column sum to the right of the window is yet to be computed for the current row, so it is centered one row above. Therefore, the first step consists of updating the column sum to the right of the window, by subtracting its topmost old pixel and adding one new pixel below it. The second step moves the window to the right and updates the window sum by subtracting its leftmost column sum (old column sum), and adding the updated column sum computed in step 1 (new column sum).

The mean filtering process is implemented in hardware with simple arithmetic operations (addition, subtraction and fixed-point multiplication) and a series of read/write operations to a memory buffer (stores the column sums) using the architecture shown in Figure 5.4 (b). The mean filter architecture receives the new pixel and the old pixel, and outputs the mean corresponding to the window being filtered. Initially, the column sum yet to be updated is read from the column sum memory (its size depends on the image width), and once updated (after adding and subtracting the new and old pixels, respectively), it is written



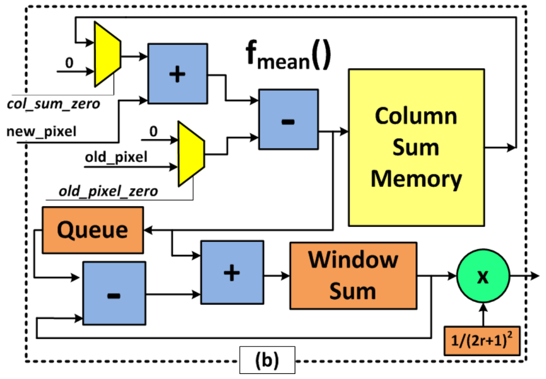


**Figure 5.4: (a) Mean Filtering Process. (b) Mean Filter Hardware Architecture**

to the memory at the same address (read operation performed one clock earlier to maintain pipeline consistency). The window sum is computed by adding and subtracting the updated and old column sums, respectively, from the content of window sum register. However, we avoid fetching the old column sum from the memory, as we aimed to make the architecture flexible for both ASICs and FPGAs supporting dual-ported BRAMs (2 ports already used to update the new column sum). Access to the old column sums is instead obtained through a shift register (queue) with size 2r+1 (an old column sum at cycle t is a delayed version of the new column sum at cycle t-2r-1). The final mean value is computed by multiplying the window sum with 1/(2r+1)2.

The architecture of the GIF is depicted in Figure 5.5. It receives two pixels from the reference image (used as guidance image) and two from the slice of the SCV to be smoothed. The architecture consists of four mean filters that compute the values of $mean_I$, $mean_p$, $corr_I$ and $cor_{Ip}$. The remaining values of the described in Algorithm 5.1 are computed using a set of arithmetic units (fixed-point multipliers, adders/subtractors). The complex division operation in step 3 of the algorithm is avoided by approximating the denominator by the nearest power of 2, thus replacing the division with a right shifter. We also avoid finding the mean values of $a$ and $b$ using the mean filter architecture of Figure 5.4 (b), as this would require high memory resources in the stereo matcher (for each disparity level and for each column: a column sum and 2r+1 pixel values need to be stored). It was found that by only computing the mean over the $x$ direction (using an accumulator and a FIFO queue) rather than on a rectangular window, it reduces the quality by less than 0.5 %, but it also eliminates the need to store the aforementioned values in BRAMs or shift registers.

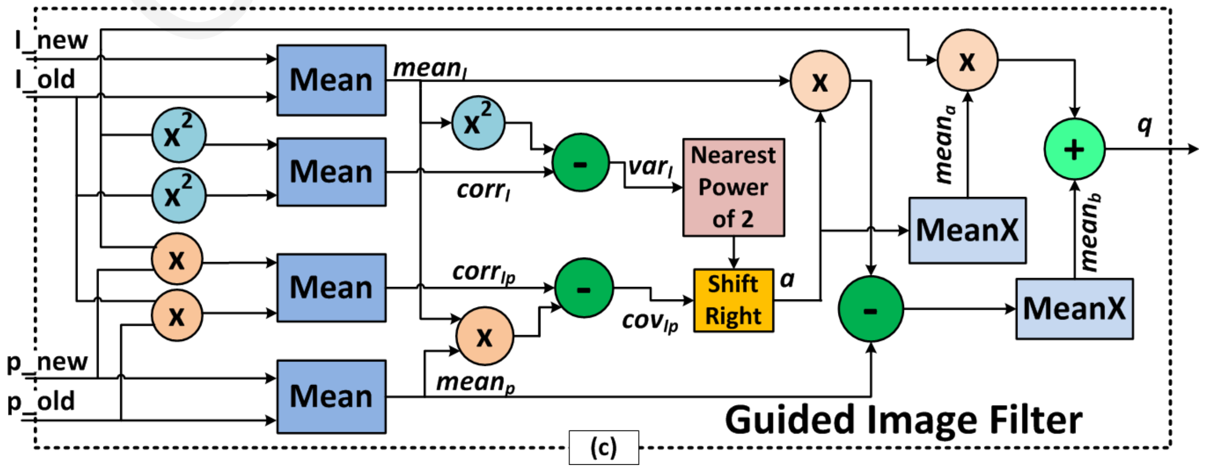


Figure 5.5: Guided Image Filter Hardware Architecture.

## 5.4 Proposed GIF-based Stereo Matcher (GIF-SM)

The edge-aware nature of the GIF, and the simplicity of its hardware architecture presented in the previous section, can benefit stereo matching dedicated hardware architectures in many ways. First of all, the filter can be utilized to perform the ADSW aggregation process in an efficient manner, in that it does not need to rely on the match window size anymore. Instead, ADSW cost aggregation can now rely on pixel-based operations. In particular, only two pixels (the *new pixel* and *old pixel*) need to be processed every clock cycle. Hence, the hardware complexity of cost aggregation can be reduced significantly (compared to the case where the weights are computed exhaustively like in the architecture presented in Chapter 4.4), or higher levels of disparity level parallelism can be exploited to enable higher frame rates. It must be noted that the architecture in Chapter 4.4 was not able to exploit parallelism along the full disparity range, whereas the GIF-based stereo matcher presented in this section can do so, thanks to the simplicity of the GIF. In addition, the GIF simplifies the design of the memory sub-system, as the memory buffers that store the RGB and gradient values can be implemented not only using shift registers, but also using FIFO/BRAM memories. Thus, the buffer types can be selected as per the design effort and application demands. It is important to note that these important benefits do not lead to a significant quality degradation of the matching quality, as the GIF performs the weighted-averaging required in ADSW aggregation, while preserving most of the edges. Last but not least, the integration of the GIF as part of the disparity refinement step to perform weighted median filtering can further improve the matching quality.

In this section, we present and discuss how the GIF can be integrated along the entire video processing pipeline of a hardware stereo matching architecture. Therefore, we will present a general architecture that integrates the GIF both for filtering the stereo cost volume (SVC) but also for weighted median filtering. In fact, the generality of the architecture lies in the “plug-and-play” nature of the SVC filtering, which may perform both an ADSW cost aggregation by integrating the GIF, or a fixed-support aggregation by simply replacing the GIF with the architecture of the mean filter (Figure 5.4 (b)). In addition, since it is possible to perform the weighted median filtering without the GIF, we also describe a general refinement unit integrating different weighted filters. Later in this chapter, we present a detailed benefit-cost analysis (in terms of matching accuracy and hardware complexity) that compares different possible configurations of the architecture (see Section 5.5.2).

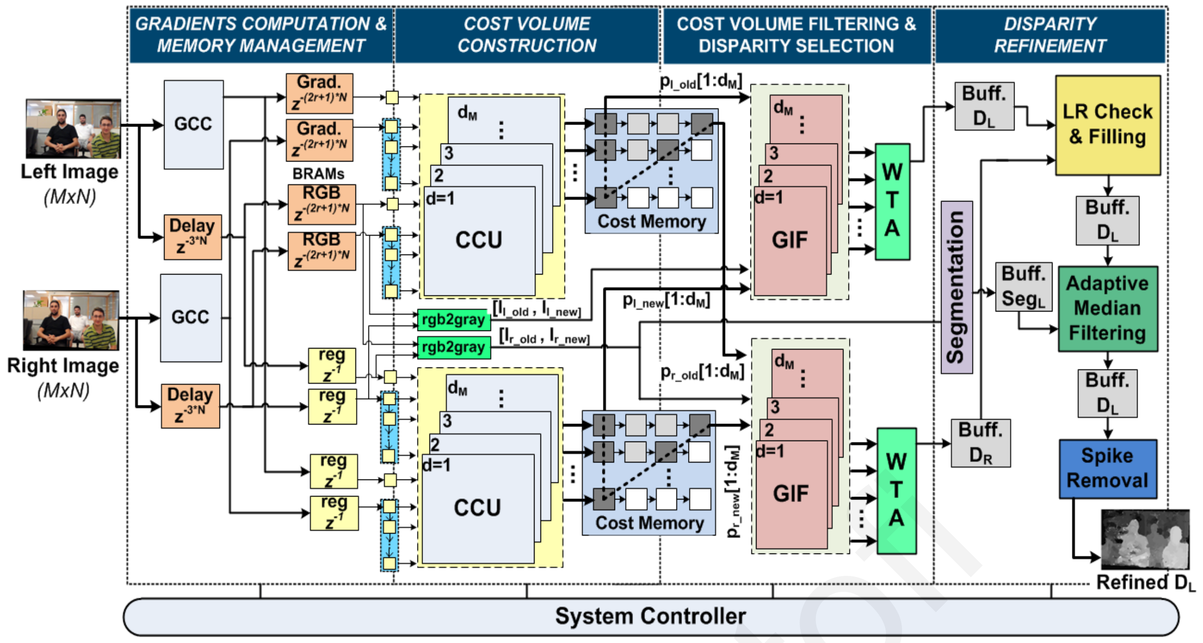


**Figure 5.6: Architecture of the proposed GIF-based Stereo Matcher.**

The proposed GIF-based Stereo Matching architecture shown in Figure 5.6 is decomposed into four major hardware units: the Gradients Computation & Memory Management Unit (GCMMU), the Cost Volume Construction Unit (CVCU), the Cost Volume Filtering & Disparity Selection Unit (CVFDSU) and the Disparity Refinement Unit (DRU). These hardware units are fully pipelined in order to obtain best processing throughput. Source pixels come progressively in scanline order from the input port (stereo camera or external memory), enter the processing pipeline consisting of the four major units mentioned above, and after a pipeline latency, refined disparities are forwarded successively in scanline order to the output port, synchronized with the input pixel rate. In this way, the architecture complies with existing pixel feeding standards Figure 5.6 shows a block diagram of the proposed architecture and the data flow between units. All the processing units and dataflow are controlled by the system controller.

## 5.4.1 Gradients Computation & Memory Management Unit (GCMMU)

The GCMMU utilizes two Gradient Computation Cores (GCC) that calculate the gradients of the input stereo images. While the original algorithm in [**142**] uses only the gradients in $x$ direction in the final SCV function, this work utilizes the gradients in $y$ direction as well, as it was found to yield better quality results. Therefore, the GCCs implement the Sobel operator to calculate the gradient values in both directions using the architecture shown in Figure 5.7 (a). The architecture performs convolution of $3x3$ windows (fetched from a scanline buffer) with the Sobel kernels using two convolution units (CONV), and normalizes the result in the range [0,255] (NORM units). The GCMMU is also responsible for buffering the computed gradients and input color values, and providing them in a synchronized way to the CVCU. In particular, it stores $2r + 1$ lines of both RGB and gradient data in single-ported BRAMs working in read-first mode. This allows access on both the new and old pixels; new data written to the BRAM represents the new pixel, while the previous content of the address put on the data output is the old pixel. The new pixels (BRAM inputs) are synchronized with the old pixels (BRAM outputs) by introducing one cycle delay through register buffers. Finally, the RGB and gradient values of the $2d_M$ most recently fetched old and new pixels in the target image (right) are stored in serial-in parallel-load shift registers. This allows the CVCU to exploit disparity level parallelism.

### 5.4.2 Cost Volume Construction Unit (CVCU)

The CVCU employs a cascade of Cost Computation Units (CCUs) that calculate the pixel-wise costs between the new and old pixels in the left image and their 2dM corresponding pixels in the right image. The architecture of a CCU is shown in Figure 5.7 (b). It consists of absolute difference units, adders and comparators that calculate the truncated color and gradient costs, which are then summed up to compute the overall cost. However, prior to the summation, the truncated color and gradient costs are multiplied by constant values, in order to balance their influence in the overall cost. The constants are selected to be powers of 2 to replace multiplication with shifting. The final pixel costs calculated by using the left image as reference are stored in cost memory buffers, and are reused for the computation of the right disparity map by the CVFDSU.

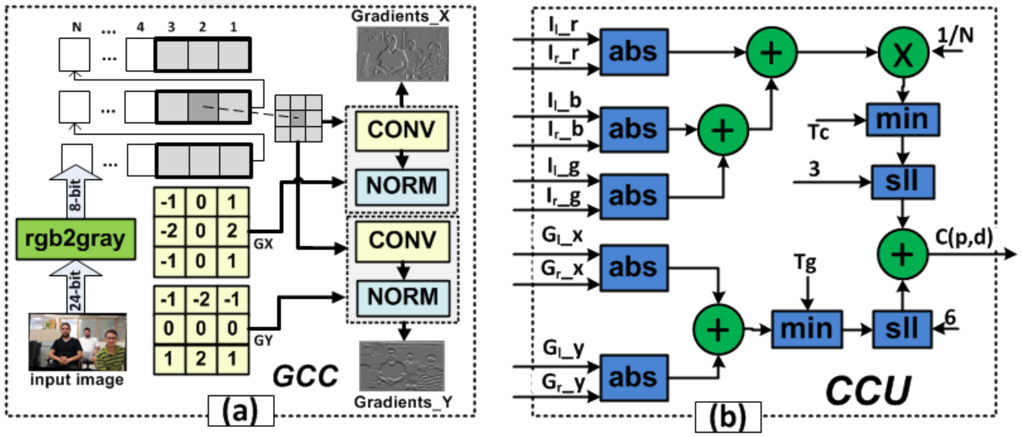


Figure 5.7: (a) Gradients Computation Core, (b) Cost Computation Unit.

### 5.4.3 Cost Volume Filtering & Disparity Selection Unit (CVFDSU)

The CVFDSU employs two series of GIFs; one working on the vertical and one on the diagonal direction on the cost memories, to smooth the SCVs corresponding to the left and right disparity maps, respectively. Smoothing is performed using the architecture of the GIF shown in Figure 5.5. However, it must be noted that the mean and covariance images of the input grayscale images are computed only once and shared between the different GIFs, in order to effectively reuse the computed data and avoid extra hardware resources. The CVFDSU also incorporates 2 WTA units (composed of comparators organized in tree structures) that select the disparities with the minimum costs. Figure 5.8 shows the architecture of the CVFDSU in detail; note that the computation of the second disparity map used during the left-right consistency check in the refinement step is computed by duplicating this architecture. As can be observed, multiple slices of the SVC can be filtered in parallel, depending obviously on the targeted FPGA and the available hardware resources.

### 5.4.4 Disparity Refinement Unit (DRU)

The DRU implements the L-R check and filling (interpolation) approach described in Section 5.2.2 through a set of comparators and priority encoders that locate the nearest valid disparities in the left and right direction of the invalid pixels. The interpolated disparity map is then filtered to reduce “salt-and-pepper” like noise and remaining spikes by weighted median

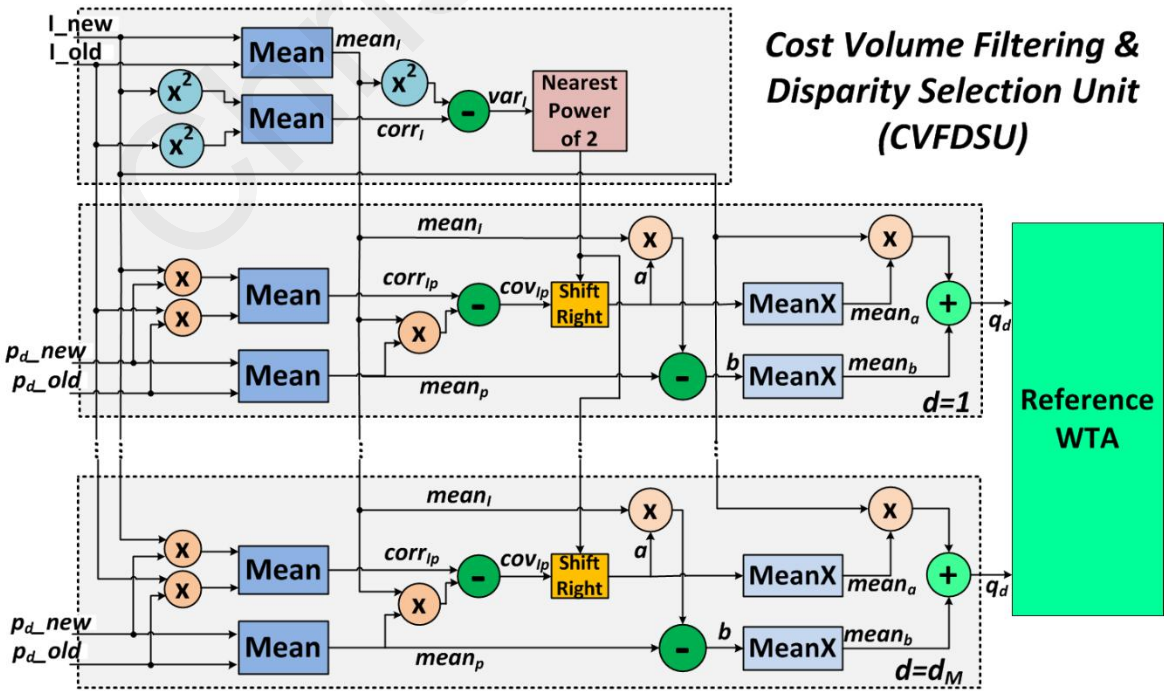


Figure 5.8: Block diagram of the CVFDSU.

filters. One approach to perform weighted median filtering is based on the segmentation-based adaptive smoothing filter presented in the previous Chapter. As a reminder, this architecture relies on cumulative histograms, to find the weighted median value, has a parallel implementation able to computation of the median value for a window of size $mxm$ in a single clock cycle. Its major hardware units include ROM memories, adder-trees, comparators and a priority encoder that locates the median value among the totality of the bins. Moreover, the binary weights generated by image segmentation ("one" if a disparity value lies in the same segment with the central pixel of the window, "zero" otherwise). This approach is not only hardware friendly, but also preserves object borders since pixels lying in the same segment are more likely to lie at the same disparity. A simple method that partitions the disparity image into segments based on thresholding was utilized. The binary weights are used as the enable signals on the ROMs.

As we discussed in the theory section in this Chapter, the GIF can also be employed to perform weighted median filtering in an efficient manner by relying on a cascade on mean filters. The same architecture used to perform smoothing of the SVC (shown in Figure 5.8) can be utilized to perform the GIF-based weighted median filtering. The only difference is that the incoming values to the architecture do not come from the slices of the SVC but are generated from the disparity map by utilizing a decoder, the size of which depends on the disparity range (e.g. 6-to-64 decoder for $d_M$=64). In fact, the decoder generates the 3D signal $f(p_c, i)$ in

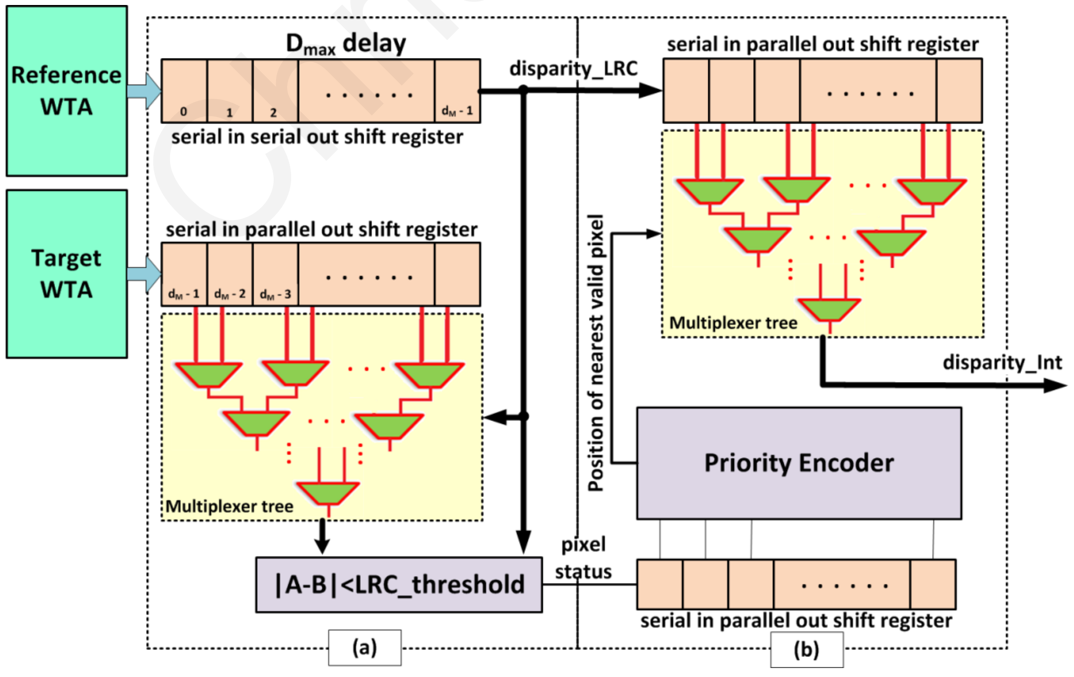

Reference WTA
Target WTA
D_max delay
0
1
2
d_M - 1
serial in serial out shift register
serial in parallel out shift register
d_M - 1
d_M - 2
d_M - 3
Multiplexer tree
|A-B|<LRC_threshold
disparity_LRC
serial in parallel out shift register
Multiplexer tree
disparity_Int
Position of nearest valid pixel
Priority Encoder
pixel status
serial in parallel out shift register
(a)
(b)

**Figure 5.9: (a) Left-Right Consistency Check, (b) Filling of invalid pixels.**

(5.17), by decoding each disparity value into $d_M$ binary values, which form the inputs of the architecture Figure 5.8. In the case, of weighted filtering, this architecture consumes less resources, due to processing binary instead of 8-bit values (as in the case of the SVC).

The DRU finally implements a typical median filter (spike removal), which is also implemented based on the cumulative histogram approach and the architecture discussed in Chapter 4.4.3, Figure 4.5; however, since the typical median filter has not weights, the enable signals of the ROMs are set to 1.

## 5.5 FPGA Implementation Results

This section explores the benefits of integrating the GIF in stereo matching hardware designs by evaluating different configurations of the generic architecture presented in the previous section using FPGA implementation and emulation. In particular, a detailed performance analysis is performed to determine the configuration that better exploits the GIF (in terms of obtaining the best accuracy/hardware overheads tradeoff for a given throughput constraint), and based on that configuration, the section provides comparisons with related dedicated hardware and software implementations.

### 5.5.1 Experimental Platform

The proposed hardware design of the GIF and the stereo matching architectures that integrate the filter design have been implemented on the Inrevium's Kintex-7 FPGA Display Kit [**133**], which is equipped with a Xilinx Kintex-7 FPGA (XC7K325T-FFG900) [**134**]. We used a custom-built stereo camera system consisting of two Sony Handycam video cameras configured to output 1280x720 images at 60 fps (720p@60Hz HD video). The stereo camera setup was directly interfaced to the FPGA board through capturing FMC daughter cards [**135**], and was utilized to capture stereo video sequences, which were rectified (through a custom hardware stereo image rectification unit that follows the architecture presented in [**145**]), and used as input data to the system architecture(s) shown in Figure 5.6. We configured the different system architecture(s) to receive stereo video sequences synchronized with the HDMI pixel sampling clock. The refined disparity maps were also synchronized with the pixel clock, and directed to an HDMI-compatible monitor. The experimental testbed and system diagram of the prototype FPGA implementation is given in Figure 5.10 (a). The resulting disparity maps for both benchmark and real-world stereo images are shown in Figure 5.10 (b) & (c), respectively.

### 5.5.2 Cost-Benefit Analysis

As already discussed in the previous sections, the GIF can be utilized both for cost volume filtering in the cost aggregation step, and also for weighted median filtering during the disparity refinement step. By integrating it in both of these processing steps, it is more likely to enable higher disparity map accuracy; however, at the cost of more hardware and power demands. In order to determine how the GIF can best benefit stereo matching dedicated hardware designs, a number of different configurations of the generic architecture of Figure 5.6 were explored. Given a targeted throughput constraint of 60 fps (imposed by the frame rate of the cameras used), this exploration aims primarily to find an effective tradeoff between matching accuracy and required hardware resources.

Table 5.1 lists the various possible system configurations. The first configuration utilizes the GIF for filtering both the SVC and the resulting disparity map (weighted median filtering). This corresponds to an ADSW stereo matching architecture. A second configuration replaces the GIF with a series of mean filters (box filtering), which is equivalent to performing the

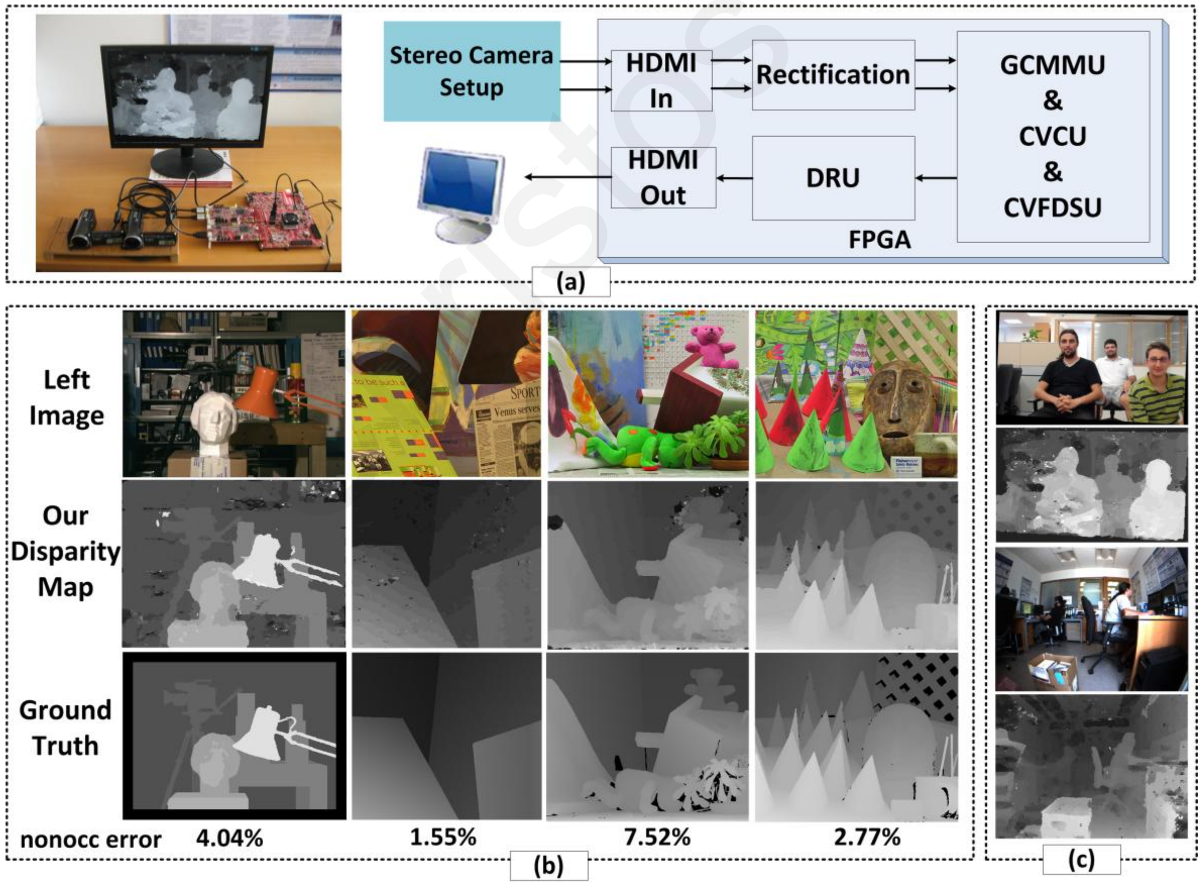


**Figure 5.10: (a) Experimental Testbed. (b) Benchmark Stereo Images. (c) Real-world Stereo Pairs Captured in the Lab.**

**Table 5.1: Definition of the various stereo matching designs that utilize the GIF.**

| System Conf. | Description | System Parameters |
|---|---|---|
| **SC1** | GIF_AGGR.[1] & GIF_WMF[2] & SB_WMF[3] | $\{r_{aggr}, \varepsilon, T_c, T_g, r_{GIF_{WMF}}, r_{SB\_WMF}\} = \{3, 0, 7, 2, 19, 9\}$ |
| **SC2** | GIF_AGGR. & GIF_WMF | $\{r_{aggr}, \varepsilon, T_c, T_g, r_{GIF_{WMF}}, r_{SB\_WMF}\} = \{3, 0, 7, 2, 19, 9\}$ |
| **SC3** | GIF_AGGR. & SB_WMF | $\{r_{aggr}, \varepsilon, T_c, T_g, r_{GIF_{WMF}}, r_{SB\_WMF}\} = \{3, 0, 7, 2, 19, 9\}$ |
| **SC4** | BOX_AGGR.[4] & GIF_WMF & SB_WMF | $\{r_{aggr}, \varepsilon, T_c, T_g, r_{GIF_{WMF}}, r_{SB\_WMF}\} = \{3, 0, 6, 1, 19, 9\}$ |
| **SC5** | BOX_AGGR. & GIF_WMF | $\{r_{aggr}, \varepsilon, T_c, T_g, r_{GIF_{WMF}}, r_{SB\_WMF}\} = \{3, 0, 6, 1, 19, 9\}$ |
| **SC6** | BOX_AGGR. & SB_WMF | $\{r_{aggr}, \varepsilon, T_c, T_g, r_{GIF_{WMF}}, r_{SB\_WMF}\} = \{3, 0, 6, 1, 19, 9\}$ |

[1] ADSW cost aggregation using the GIF; [2] Weighted median filtering using the GIF;
[3] Segmentation based weighted median filtering; [2] Fixed-support aggregation using box filtering;

aggregation using a fixed support window (due the use of equal weights). In this case, the required hardware resources should be much lower; instead of using two series of cascaded GIFs (that are implemented using cascades of mean filters), aggregation is perform by utilizing only a cascade of mean filters (considering the symmetrical nature of this aggregation type). The problems of fixed support aggregation are well discussed several times in this thesis. This configuration aims to investigate whether integrating the edge-aware GIF in the refinement step can alleviate some of these problems. The rest of the configurations listed in the table result from different combinations of weighted median filters (e.g. the GIF with the segmentation-based weighted median filter presented in Chapter 4, or each filter alone). The table also lists the different system parameters associated to each configuration; these parameters were found empirically to yield the maximum matching quality for the specific configuration. Figure 5.11 provides a visual illustration of the possible configurations (dashed lines indicate the units that take part in the different combinations).

The various stereo matching designs listed in Table 5.1 are compared in terms of matching accuracy (measured in terms of the percentage of bad matching pixels relative to ground truth disparity maps from the Middlebury dataset [**26**]), and FPGA resource usage (Slice LUTs, Slice Registers, embedded memory and DSPs). These metrics are analyzed based on the minimum clock frequency required to fulfill the fixed throughput constraint (60 fps)

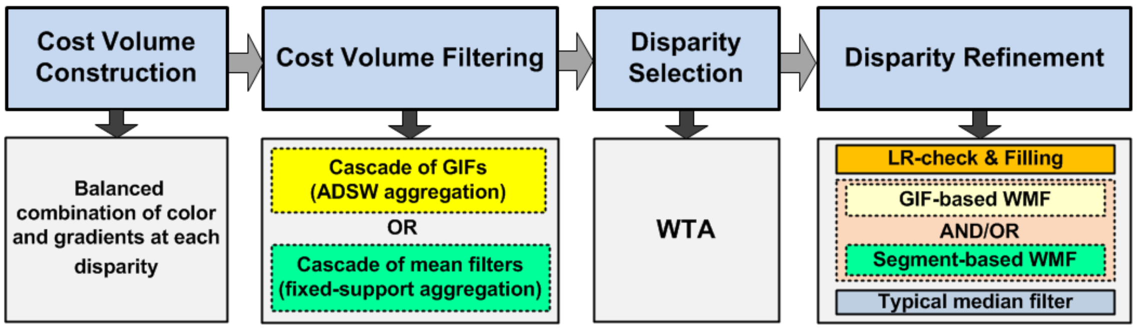


**Figure 5.11: Various Stereo Matching Hardware Designs utilizing the GIF.**

**Table 5.2: FPGA resource usage of the various system configurations.**

| System Configuration | Slice LUTs | | Slice Registers | | DSP48E | | BRAM | |
|---|---|---|---|---|---|---|---|---|
| | *Total=203800* | | Total=407600 | | Total=840 | | Total=445 | |
| **SC1** | 181866 | ~89% | 139704 | ~34% | 738 | ~87% | 441 | ~99% |
| **SC2** | 179015 | ~87% | 135303 | ~33% | 738 | ~87% | 437 | ~98% |
| **SC3** | 97287 | ~47% | 129964 | ~31% | 738 | ~87% | 271 | ~60% |
| **SC4** | **55739** | **~27%** | **38303** | **~9%** | **328** | **~39** | **310** | **~69%** |
| **SC5** | 52888 | ~25% | 33902 | ~8% | 328 | ~39 | 306 | ~68 |
| **SC6** | 36396 | ~17% | 29842 | ~7% | 4 | ~1 | 140 | ~31 |

imposed by the stereo cameras; this frequency is 74.25 MHz for a camera resolution of 1280x720 (720p HD video). The FPGA utilization figures and matching quality are summarized in Table 5.2 & Table 5.3, respectively. As can be observed, SC1 is associated with the lower error matching rate, but at the same time, it results in the most hardware demanding design. On the other hand, the most efficient hardware design in terms of utilized FPGA resources is SC6. However, this design, which implemented simple box aggregation with fixed window size, produces the less accurate disparity maps.

To compare the different hardware implementations in terms of matching accuracy and FPGA resource usage, we developed tradeoff x-y scatter plots for each different type of FPGA resource (i.e. LUTs, registers, etc.) versus the average error matching rate. The scatter plots are depicted in Figure 5.12. Ideally, the configurations yielding the better accuracy/resource usage tradeoffs should be located at the right lower corners of the scatter plots, suggesting that SC4 and SC5 are the best possible choices. These configurations integrate the GIF in the disparity refinement step, and perform weighted median filtering on the disparity map by utilizing only a cascade of GIFs that process binary values (see equation (5.17)). The cost aggregation step is perform with simple box filtering, which due to its symmetry, it requires only filtering of a single SVC with a cascade of box (mean) filters; the values of the filtered SVC are reuse for computing the other filtered SVC.  Hence, SC4 and SC5 consume less hardware resources compared to the most accurate configuration (SC1), which however, filters two independent

**Table 5.3: Matching accuracy of the different system configurations.**

| System Conf. | Tsukuba | | | Venus | | | Teddy | | | Cones | | | Avg. Error |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *NONOCC* | *ALL* | *DISC* | *NONOCC* | *ALL* | *DISC* | *NONOCC* | *ALL* | *DISC* | *NONOCC* | *ALL* | *DISC* | *%* |
| **SC1** | 2.57 | 3.09 | 8.89 | 0.30 | 0.52 | 2.65 | 6.96 | 12.3 | 16.8 | 2.91 | 8.59 | 8.48 | 6.17 |
| **SC2** | 3.42 | 3.99 | 9.72 | 0.48 | 0.66 | 3.29 | 7.05 | 12.4 | 16.0 | 3.11 | 8.22 | 9.01 | 6.45 |
| **SC3** | 4.78 | 5.28 | 9.47 | 1.86 | 2.22 | 5.54 | 8.09 | 13.5 | 18.1 | 3.36 | 9.37 | 9.32 | 7.57 |
| **SC4** | **2.38** | **3.01** | **9.38** | **0.40** | **0.70** | **3.62** | **7.23** | **12.7** | **17.2** | **2.87** | **8.59** | **8.27** | **6.36** |
| **SC5** | 3.48 | 4.11 | 10.3 | 0.89 | 1.21 | 5.09 | 7.41 | 12.8 | 16.7 | 2.86 | 8.64 | 8.26 | 6.81 |
| **SC6** | 5.17 | 5.92 | 11.0 | 2.93 | 3.47 | 8.01 | 9.05 | 14.5 | 20.3 | 3.81 | 9.94 | 10.2 | 8.70 |

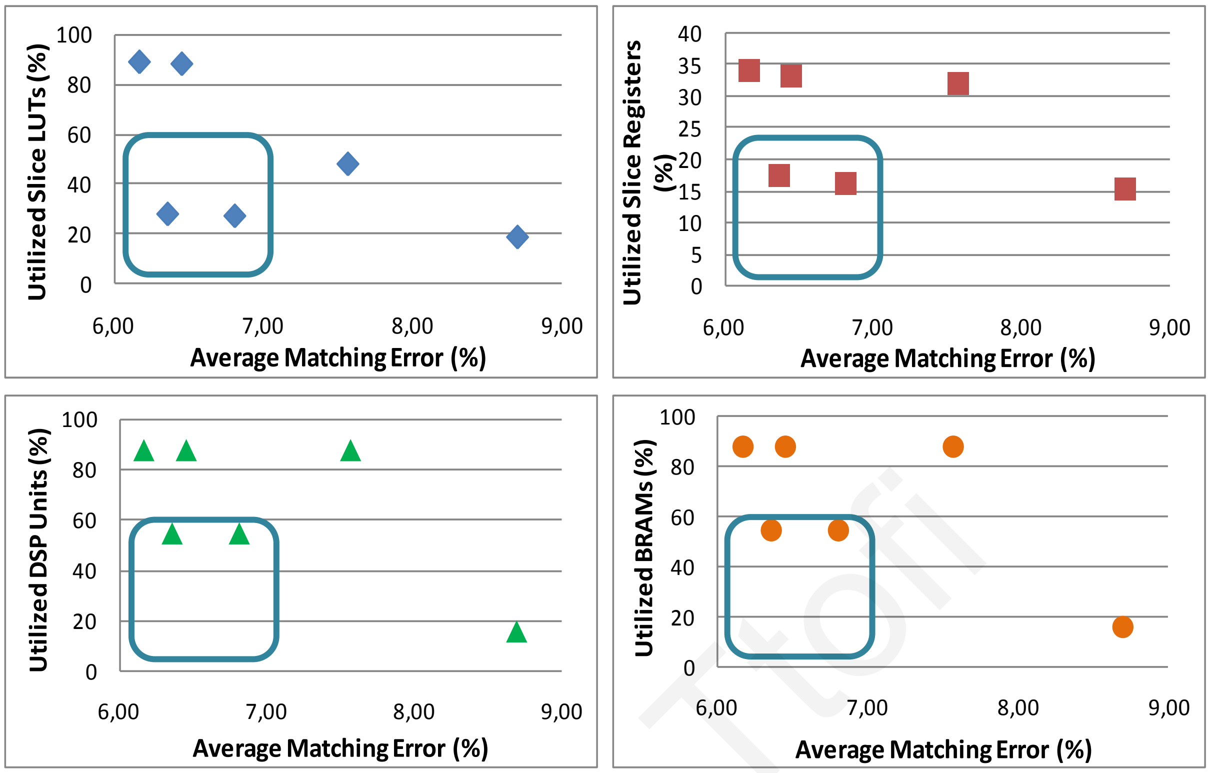


**Figure 5.12: FPGA resource usage / matching accuracy tradeoff analysis using scatter plots.**

SVC (one when using the left images as reference, and one when using the right.) through two series of cascaded GIFs, thus consuming excessive hardware resources.

Among configurations SC4 and SC5, we selected the former, as it is associated with lower matching error rate (due to the fact that it also integrates the segment-based weighted filter). Despite the significant savings in hardware resources when implementing SC4 over the most accurate configuration SC1, the former configuration achieves a matching quality that is on par with the quality of SC1. The general conclusion extracted from the aforementioned tradeoff analysis is that disparity refinement can be at least as crucial as the so overlooked cost aggregation. In addition, the GIF-based weighted median filter, thanks to its edge-aware nature and compact hardware design, can significantly improve the matching quality even when it is combined with simple box aggregation strategies. This is particularly important for resource constrained embedded vision systems that require high-quality depth computation in real time.

### 5.5.3 Comparisons with Related Work

#### 5.5.3.1 *Matching Quality*

We used the Middlebury benchmark dataset [**26**] to evaluate the quality of the resulting disparity maps. We measured the percentage of bad matching pixels relative to the ground truth disparity maps of four pairs of test images in the dataset. The benchmark disparity maps and those produced by the selected system configuration are shown in Figure 5.10 (b). Quantitative evaluation results are listed in Table 5.4 (columns 2-5), which also provides a comparison with related work. As it can be observed, the proposed GIF-based architecture generates high quality results, even at regions with low texture and close to object boundaries; this is evidenced by obtaining the lowest error rates among related hardware implementations at *DISC* regions. Most importantly, among the implementations for which an overall percentage of bad matching pixels is provided, the proposed system obtains the lowest error rate. Besides the Middlebury benchmark images, the proposed architecture is able to deal with real-world scenes as well, producing detailed and accurate disparity maps with clean object boundaries (see Figure 5.10 (c)).

We have also investigated how the resulting disparity maps are compared with those generated by hybrid CPU/GPU implementations [**142**], [**143**] that adopt GIF-based stereo matching algorithms. As it can be observed from Table 5.4, the distance in quality between the works in [**142**], [**143**] and the proposed architecture is only 0.84% and 0.17% on average. It should be noted, that these implementations utilize a color version of the GIF, which in general yields better accuracy. Therefore, the distance in quality can be further reduced if the proposed GIF-SM is extended to support color guidance images as well. Finally, it is worth mentioning that the matching accuracy of the proposed GIF-based stereo matcher (SC4) is slightly better than the quality of the original ADSW algorithm in [**62**] (6.36% vs. 6.67%), while the ADSW dedicated hardware implementations in [**115**], [**68**] exhibit a quality reduction compared to [**62**] of ~4.84% and ~4.68%, respectively. This evidences the superiority of the proposed hardware-based GIF-SM in maintaining the matching accuracy of the original ADSW algorithm, thanks to the integration of the GIF.

### 5.5.3.2 *Processing Speed*

We measured the processing speed of the proposed GIF-SM in frames processed per second (fps) and in Million Disparity Estimations per second (MDE/s), a metric that also takes into account the number of pixels and the disparity range (MDE/s=M·N·D·fps). The FPGA prototype of the architecture (SC4) is able to process 720p (1280x720) HD video at 60 fps, with a pixel clock rate of ~74.25MHz. When considering the post place-and-route frequency (103 MHz) provided by the Xilinx ISE Design tool, the maximum throughput of the architecture is expected to reach ~83 fps. In general, the architecture presents good scalability with respect to the frame rate and image size, as it is intensively pipelined and synchronized with the pixel clock rate. Therefore, it can easily be configured to process Full HD 1080p video (pixel clock = 145 MHz) by simply increasing the number of pipeline stages.

Table 5.4 (columns 6-10) presents a comparison between the developed FPGA prototype and related work in terms of processing speed. Obviously, the proposed system is among the fastest implementations when considering the MDE/s metric, outperforming all dedicated hardware implementations targeting high-quality disparity map estimation. Only the work in [13] obtains higher MDE/s. However, this implementation is not competitive in terms of quality, mainly due to the use of a fixed support algorithm and its associated problems discussed in Chapter 2.5.1.2. Finally, the proposed system has a speedup improvement of ~17x and ~8x when compared to the hybrid CPU/GPU GIF-based stereo matching systems in [**142**] and [**143**], respectively, thus justifying the very small quality reduction. Conclusively, the obtained speed/quality results indicate that the proposed system has high potential for embedded vision applications requiring fast and accurate depth computation.

### 5.5.4 FPGA Synthesis Results - System scalability

Table 5.5 summarizes synthesis results in terms of FPGA utilization figures, operating frequency and power consumption for each of the major components of the selected GIF-based Stereo Matcher (system configuration 4 – SC4 in Table 5.1), in order to give a complete picture of the required hardware overheads associated with the system. The complete FPGA prototype of this system for a disparity range of 64 pixels ($d_M = 64$) utilizes ~28% of the available Slice LUTs and ~18% of the available Slice Registers. It also utilizes ~55% of the

**Table 5.4: Quality and Processing Speed Comparison with Related Work.**

| Work | Average Error Rates | | | | Image size | D | Speed (fps) | MDE/s ($10^6$) | Platform |
|---|---|---|---|---|---|---|---|---|---|
| | *NONOCC*[1] | *ALL*[2] | *DISC*[3] | *OVERALL*[4] | | | | | |
| **Baha [137]** | 7.16 | 11.5 | n.a. | n.a. | 450x375 | 50 | 46 | 388 | FPGA |
| **Zhang [140]** | 4.41 | 7.41 | 12.8 | 8.20 | 1024x768 | 64 | 60 | 3019 | FPGA |
| **Jin [108]** | 8.31 | 13.9 | 27.2 | 16.5 | 640x480 | 64 | 230 | 4522 | FPGA |
| **Ambrosch [109]** | 5.82 | 9.37 | 22.2 | 12.5 | 750x400 | 60 | 60 | 1080 | FPGA |
| **Chang [114]** | n.a. | 6.81 | n.a. | n.a. | 352x288 | 64 | 42.5 | 276 | ASIC |
| **Ding [115]** | 7.41 | 11.9 | 15.6 | 11.5 | 640x480 | 60 | 51 | 940 | FPGA |
| **Perri [68]** | 4.39 | 10.09 | 19.57 | 11.35 | 640x480 | 60 | 45 | 829 | FPGA |
| **Gehrig [138]** | n.a. | 8.13 | n.a. | n.a. | 340x200 | 64 | 27 | 118 | FPGA |
| **Banz [139]** | 8.43 | n.a. | n.a. | n.a. | 640x480 | 128 | 30 | 1179 | FPGA |
| **ADSW [62]** | 3.48 | 6.53 | 9.98 | 6.67 | 320x240 | 30 | 0.01 | 0.0263 | n.a. |
| **Hosni [142]** | 2.65 | 5.57 | 8.43 | 5.55 | 640x480 | 26 | 25 | 200 | CPU/GPU |
| **Ma [143]** | n.a. | n.a. | n.a. | 6.19 | 450x375 | 59 | 45 | 448 | CPU/GPU |
| **Proposed (SC4)** | 3.22 | 6.25 | 9.61 | 6.36 | 1280x720 | 64 | 60 | 3538 | FPGA |

[1]all points except for occluded areas, [2] all points including half-occluded regions, [3] only points along depth discontinuities. [4] average error rate at *nonocc*, *all* and *disc* regions.

**Table 5.5: Kintex-7 Synthesis Results for major system components.**

| **Design Unit** | **Slice LUTs** | | **Slice Registers** | | **DSP48E** | | **BRAM** | | **Freq.** | **Power** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | ***Total=203800*** | | **Total=407600** | | **Total=840** | | **Total=445** | | **MHz** | **Dynamic** | **Static** |
| **Rectification** | 3473 | 1.7% | 6146 | 1.5% | 0 | 0% | 24 | 5.4% | - | - | - |
| **BOX_AGGR-16** | 8023 | 3.9% | 5861 | 1.4% | 34 | 4% | 56 | 12.6% | 181 | 0.5 | ~0.13 |
| **BOX_AGGR-32** | 14943 | 7.3% | 10085 | 2.5% | 66 | 7.9% | 72 | 16.2% | 181 | 0.7 | ~0.13 |
| **BOX_AGGR-64** | 29401 | 14.4% | 49702 | 12.2% | 130 | 15.5% | 104 | 23.4% | 181 | 1.3 | ~0.13 |
| **LR-check & Filling** | 504 | <1% | 650 | <1% | 0 | 0% | 0 | 0% | 489 | 0.08 | ~0.03 |
| **SB_WMF** | 2851 | 1.4% | 4401 | 1% | 0 | 0% | 4 | <1% | 500 | 0.27 | ~0.13 |
| **UMF** | 1718 | <1% | 1778 | <1% | 0 | 0% | 0 | 0% | 679 | 0.02 | ~0.13 |
| **GIF_WMF-16** | 5527 | 2.7% | 2821 | 0.7% | 84 | 10% | 50 | 11.2% | 103 | 0.33 | ~0.13 |
| **GIF_WMF-32** | 10131 | 4.9% | 4701 | 1.2% | 164 | 19.5% | 90 | 20.2% | 103 | 0.61 | ~0.13 |
| **GIF_WMF-64** | 19343 | 9.5% | 8461 | 2.1% | 324 | 38.6% | 170 | 38.2% | 103 | 1.2 | ~0.13 |
| **GIF-SM-16 (SC4)** | 21920 | 10.7% | 21223 | 5.2% | 122 | 14.5% | 134 | 30.1% | 103 | 1.14 | ~0.13 |
| **GIF-SM-32 (SC4)** | 33570 | 16.5% | 27490 | 6.7% | 234 | 27.9% | 190 | 42.7% | 103 | 1.64 | ~0.13 |
| **GIF-SM-64 (SC4)** | 57492 | 28.2% | 71192 | 17.5% | 458 | 54.5% | 302 | 67.9% | 103 | 2.8 | ~0.13 |

BOX_AGGR-x: GCMMU & CVCU & CVFDSU for $d_M = \{16,32,64\}$
GIF_WMF-x: Weighted median filter using guided filter weights for $d_M = \{16,32,64\}$
SB_WMF: Segment-based weighted median filter
UMF: Unweighted median filter

DSP units and ~68% of the block memories. With respect to the on-chip power consumption, the entire system dissipates ~3 W. Table 5.5 also provides the hardware overheads of the proposed GIF-SM (SC4) for different values of $d_M = \{16,32,64\}$, in order to investigate how it scales with respect to $d_M$. It can be observed that the hardware implementation complexity of the GIF-SM scales almost linearly with the number of disparity levels when considering the number of utilized LUTs and DSP units. The amount of utilized slice resources exhibits a quadratic increase, which is mainly attributed to the size of the cost memory buffer aimed to store the aggregated scores that are reused for the computation of the second disparity map (the size of the buffer is $d_M/2$). Regarding the scalability of the GIF-based weighted median filter (GIF-WMF), it is observed that different FPGA resources consumed by this component increase almost linearly with the disparity range. The remaining components listed in the table, which include pre-processing (rectification) and post-processing units (LR-check & Filling and unweighted median filtering) consume relatively low FPGA resources.

## 5.6 Concluding Remarks

This chapter investigated how to integrate the edge-preserving guided filter into stereo matching hardware designs, as a means to enable high-quality and concurrently real-time depth computation. A compact and efficient hardware design of the filter was illustrated, which was then utilized to develop a number of different stereo matching architectures, implementing both fixed-support and adaptive-support weight methods. The different GIF-based stereo matching designs were compared in terms of matching quality / resource usage tradeoff, and the configuration that best exploits the benefits on the GID was selected to be implemented as part of a complete stereo vision system on an FPGA platform. Experimental results have shown that the proposed GIF-based stereo matching design can enable real-time frame-rates even on HD images. Moreover, the GIF's inherent edge-preserving nature allows for improved matching accuracy when compared to existing state-of-the-art hardware stereo matching systems.

# CHAPTER 6

# Incorporating Real-Time Depth Computation in Embedded Vision Applications

*THIS chapter aims to show the significance of real-time depth computation in embedded vision applications. The design of a depth-accelerated hardware object detection system is first introduced, following by a real-time obstacle avoidance system for mobile robots. These example applications, which utilize depth information provided by the architectures presented in the previous chapters, clearly show that the design of stereo vision architectures satisfying real-time and low-power constraints is and remains important for implementing cost-effective embedded vision systems.*

## 6.1 Depth-Accelerated Hardware Object Detection

The section provides useful insights obtained from the integration of the edge-directed stereo matching design in a hardware object detection system. Section 6.2.1 introduces the problem of object detection, its challenges, and the motivation behind this work. Section 6.1.2 gives a summary on the object detection process, and explains how depth information can be used to reduce the overall search space involved in this process. Section 6.1.3 provides information about related work on object detection hardware systems. Section 6.1.4 presents the proposed depth-guided object detection hardware architecture, and Section 6.1.5 presents the experimental platform and simulation results.

### 6.1.1 Introduction - Motivation

Object detection, an important used task in several computer vision and artificial intelligence applications, refers to the ability of a computer system to determine the presence of objects of interest in images. Emerging applications that utilize object detection such as human-computer interaction, surveillance, biomedical imaging, space missions, and automobile applications among others, require real-time detection and low energy consumption. In such cases, software implementations of object detection usually struggle to meet the real-time performance and low energy constraints, and as a result some form of hardware acceleration or even a complete custom hardware implementation is preferred [**146**], [**147**]. Additionally, the improved performance stemming from hardware architectures extends the application spectrum in the 3D world, where stereo and multi-view cameras can be used for object detection in 3D environments. Furthermore, the performance in terms of detection frame rates of hardware architectures enables optimization steps to be included in the algorithm, while maintaining the real-time constraints. Such steps include image enhancement techniques, segmentation and other well-known steps traditionally used in software implementations.

Several hardware-based object detection architectures yielding real time results emerged in recent works [**148**], [**147**], [**149**]. The majority of these architectures employ the traditional sliding search window approach, which generates a lot of data and makes the object detection process time consuming and tedious in terms of hardware data flow. Moreover, only a few implementations feature optimizations, which have been widely used in software implementations, that can potentially contribute in reducing the hardware constraints in terms of efficient data handling and energy consumption. Some of the few hardware optimizations employed, utilize techniques such as color segmentation and background segmentation [**150**], [**151**]. Recently, the idea to utilize depth information emerging from disparity maps in stereoscopic applications has been proposed for reducing the effective search space in the input image. Such software implementations have yield promising results [**152**], [**153**], [**154**].

Motivated by the software mechanism presented in [**152**], this section provides insights about a hardware integration of depth information extracted from stereoscopic disparity map computation, as a means to reduce the search space and energy consumption of hardware object detection systems. This reduces the overall search area of the classifier, which in turn allows real-time detection of larger images as well as reduction of the overall energy consumption, when compared to naive sliding window search used in traditional object detection. In contrast to the software implementation in [**152**], we exploit the inherit parallelism of the disparity map computation and object detection algorithms by integrating a variant of the edge-directed disparity computation architecture presented in Chapter 3 along with a hardware SVM-based object detection classifier developed from the base model presented in [**155**], to implement an optimized object detection framework for hardware stereoscopic systems. This section focuses on providing a discussion about the hardware design parameters and the necessary trade-offs involved in using the disparity information, and presents a complete experimental platform that includes a stereoscopic camera capturing platform, camera calibration, classifier training and visual display. The entire system was implemented on a Virtex-5 FPGA platform targeting the application of face detection for a stereo image pair of 240x320 (row x column) pixels. We observe frame-rate speedups in the range of 1.9-4.7 when compared to a traditional sliding window approach and estimated energy savings of 41-48%.

## 6.1.2 Background

### 6.1.2.1 *Object Detection Overview*

The process of image object detection deals with determining whether an object of interest is present in an image/video frame or not, regardless of its size, orientation, and environmental conditions. An image object detection system receives an input image/video frame, and subsequently searches to locate possible objects of interest. This search is usually done by extracting smaller regions from the frame, called search windows, of *m x n* pixels, which are processed by a classification algorithm to determine if they belong to the object of interest or not. To account for the variable sizes in objects given the typically fixed size of the search window, an object detection system usually downscales the input image in steps, effectively reducing the size of the object of interest, and reexamines the image, until the downscaled image is equal to the size of the search window. Many downscaled images are produced from a single input image/video frame, each in turn producing a number of search windows, which increases the amount of data that must be processed by the classification algorithm. While exhaustive search has been the most popular approach, the number of search windows becomes prohibitive as the size of the input image increases. Popular methods that have been used to reduce the number of generated windows include skin segmentation, background removal and recently depth information.

### 6.1.2.2 *Depth Extraction*

Depth information can be extracted by using a stereo vision system which works similarly to the way that the human visual system infers depth. Stereo vision systems can infer depth information about a scene from a stereo image pair (usually referred to as left and right images) [**9**], [**45**]. Depth information evolves from the computation of the disparity map, from which information about the depth of objects in an image frame, relative to the location of the camera(s), can be extracted. Generally, there are three important tasks for computing the disparity map: *camera calibration*, *stereo image rectification* and *stereo correspondence*. Calibration integrates information from intrinsic camera parameters such as focal length ($f$), and extrinsic parameters such as relative orientation angles between the two cameras, to determine the camera perspective projection matrices, necessary for the stereo rectification algorithm. Rectification can project one of the two images in the other's common plane to reduce a 2D search problem in non-rectified images into a 1D search problem along the horizontal raster lines of the rectified images [**45**]. The stereo correspondence algorithm takes the two rectified images as input and produces a disparity map *d(x,y)* for each pixel in one of the two images. Generally, the disparity map stores distances between corresponding points of interest in the two images, and is computed using searching and matching techniques. Depth information ($Z$) can be computed from the disparity map using the formula $Z = f * (b/d)$ where $b$ refers to the baseline distance between the stereo camera optical centers.

### 6.1.2.3 *Depth Guided Object Detection*

By using depth information extracted from a pair of stereo images it is possible to estimate the size of the object at a given location, thus avoiding the downscaling process and subsequently reducing the generation of a large number of search windows from each downscaled version. The relationship between the actual size of the object ($O_{size}$) as is represented in the real 3D world and its projection in one of the stereo image frames ($W_{size}$) can be estimated using equation (6.1) [**45**]. The equation essentially relates the size of the object in the camera image plane with its size in the actual image. Using the relationship between depth and the disparity map (d), we can use the disparity values instead (6.2), thus, avoiding the direct computation of depth.

$$(\mathrm{W_{size}}/f) = (O_{size}/Z) \rightarrow \mathrm{W_{size}} = f * (O_{size}/Z) \tag{6.1}$$

$$Z = f * (b/d) \rightarrow \mathrm{W_{size}} = (O_{size}/b) * d \tag{6.2}$$

The depth-directed search overall procedure is given in Figure 6.1. Each value in the disparity map corresponds to a certain search window size and the coordinates of that value are at the center of that window. Consequently, there is no need for exhaustive search in the disparity map on a pixel by pixel basis, but every few pixels, the same way that the sliding window operates in traditional object detection schemes. Furthermore, if a large object is found at the border of the image the window size it will most likely exceed the image dimensions. That window can then be discarded and that object is left to be detected in another

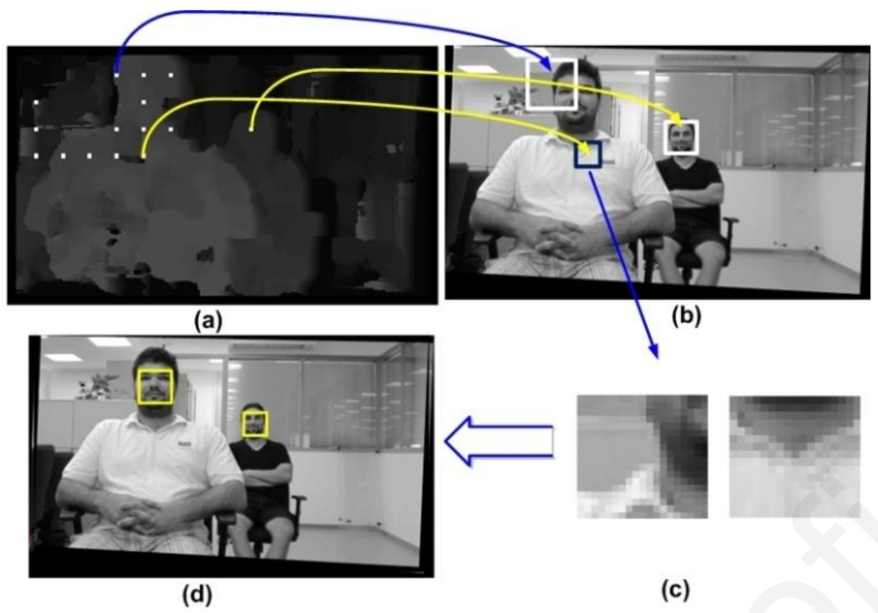


**Figure 6.1: Window Size Estimation Algorithm. (a) The disparity map is sampled every few pixels (b) For each disparity value the corresponding window size is estimated (c) Read window pixel values from one of the two stereo images (d) Output result of classified windows.**

window. Finally, any disparity values that map to windows which are smaller than the expected window size are discarded and the next disparity value is processed. In this way the number of candidate search windows is further reduced compared to the traditional object detection search method.

### 6.1.3 Related Work

In recent years, a fair amount of work has been done in hardware implementations of object detection algorithms. These works utilize techniques such as neural networks [**146**], [**148**] and the Viola Jones detection algorithm [**147**], [**149**], and achieve relatively high detection frame rates. However, the majority of these implementations follow the sliding window search approach, where as mentioned earlier, a lot of windows investigated do not contain object information, and as such a large amount of energy is wasted. Moreover, as the input image size increases, this problem is further emphasized.

The use of optimization techniques, such as skin color segmentation and background removal as a means to reduce the number of windows to be processed, and increase the resulting frame rate, is suggested in several software implementations [**150**], [**156**], [**157**], with only a few hardware implementations reported thus far [**146**], [**158**]. The former technique, however, can only be applied to applications related to skin-based detection such as face detection, while the latter may have increased complexity and computational demands depending on the detection scenario. Alternatively, depth information as a means to reduce the search space has recently been proposed in [**152**]. Depth is interpreted in the same way in many environments and thus provides a more efficient way to reduce the search space for different applications. Depth information can also be used after the detection process to reduce the false positive detections, as shown in [**153**]. The work done in [**153**], however, targets improved classification accuracy rather than reducing the search space to increase the performance.

To the best of our knowledge, this is the first attempt that investigates the use of depth as search reduction mechanism in hardware for performance improvement and energy reduction. There has been some initial work done in software [**152**], [**150**], [**154**]. The work done in [**152**] demonstrated the performance speedups from reducing the search space. The authors developed a methodology that uses depth information to estimate the size of each search window, which is then fed to a Viola-Jones face detector. However, the approach presented in that work is only suitable for video sequences where the scene does not change much between successive frames, otherwise the frame rate drops significantly. Moreover, the Viola-Jones classifier requires many hardware resources [**147**] so a more flexible and simple classifier might be appropriate. In this work, we attempt to optimize the depth based algorithm and implement it in hardware, to allow for scalability and performance optimizations, and potentially energy reduction. The depth-directed hardware implementation presented here attempts to search a larger part of the depth information compared to [**152**] by utilizing parallel hardware accesses to the depth information memory. We integrate the depth estimation approach into an existing object detection hardware implementation, in order to explore the potential performance and energy consumption benefits.

### 6.1.4 Depth-Accelerated Object Detection Hardware Architecture

The depth-accelerated object-detection architecture consists of three major hardware units; the Disparity Extraction Unit (DEU), the Window Extraction Unit (WEU) and the Classification Engine (CE). The system also consists of a system controller that optimizes accesses to the external memory, controls I/O operation, and synchronizes the other major units. The system uses two on-chip buffers to temporarily store the image data; the first buffer stores the disparity values used for the search, and the second buffer stores the parts of the image that are being searched for potential objects by the CE. Figure 6.2 shows an overview of the system architecture and the communication flow between units.

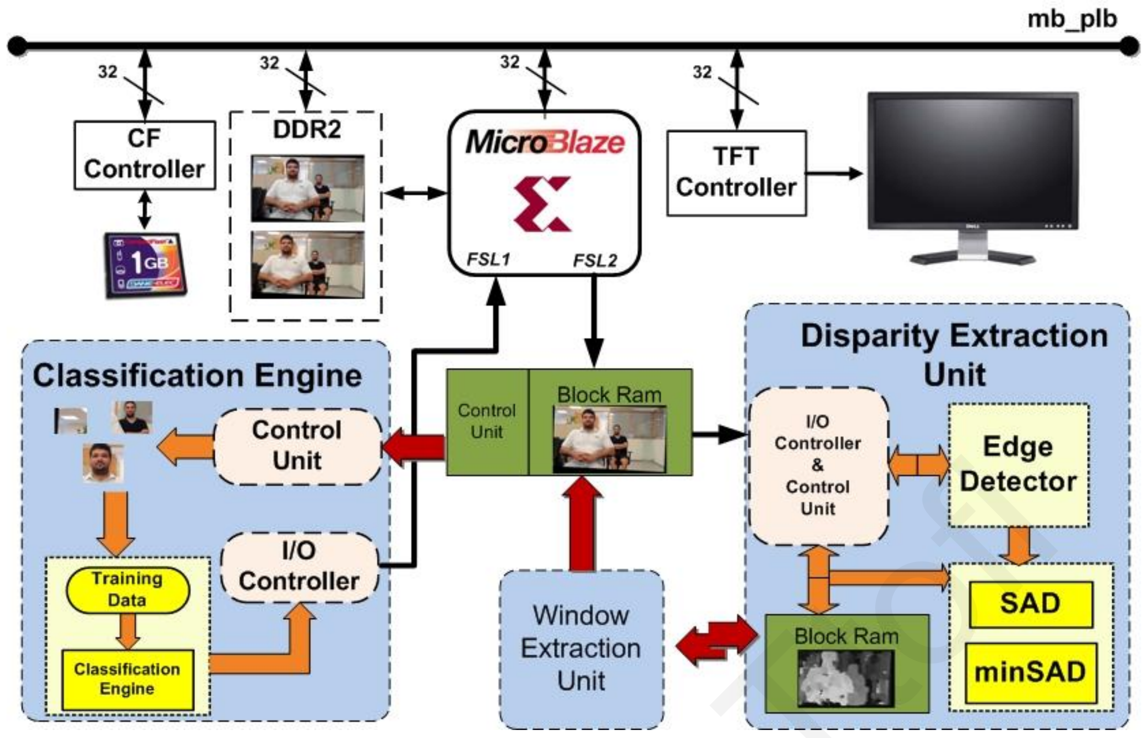


Figure 6.2: Proposed Hardware System Architecture.

### 6.1.4.1 *Disparity Extraction Unit*

The Disparity Extraction Unit (DEU) integrated to the system is based on the edge-directed disparity estimation hardware architecture presented in Chapter 3. Therefore, the DEU combines the SAD matching algorithm with the features (edges) extracted by the use of an edge detector in order to perform correlation on rectified stereo image pairs that present a maximum disparity range of 40 pixels (maximum correlation window sizes up to 11x11). The DEU uses optimized memory accesses to the external memory and was designed with emphasis on parallelism. These features enable the architecture to obtain frame rates that exceed the 150 fps for an image size of 240x320. This is particularly important, as the time for disparity computation must be small enough in order to obtain a speedup in the overall operation (disparity computation + object detection).

### 6.1.4.2 *Window Extraction Unit*

The window extraction unit (WEU), shown in Figure 6.3, is responsible for computing (6.2) which estimates the size of the disparity-directed search window, based on the disparity values provided by the DEU. It essentially links the DEU with the classification engine by generating the addresses for the pixels that will be fed to the classification engine. The WEU receives disparity values and the corresponding coordinates of those values from the DEU. It then estimates the necessary search window size in the original input image. The classification

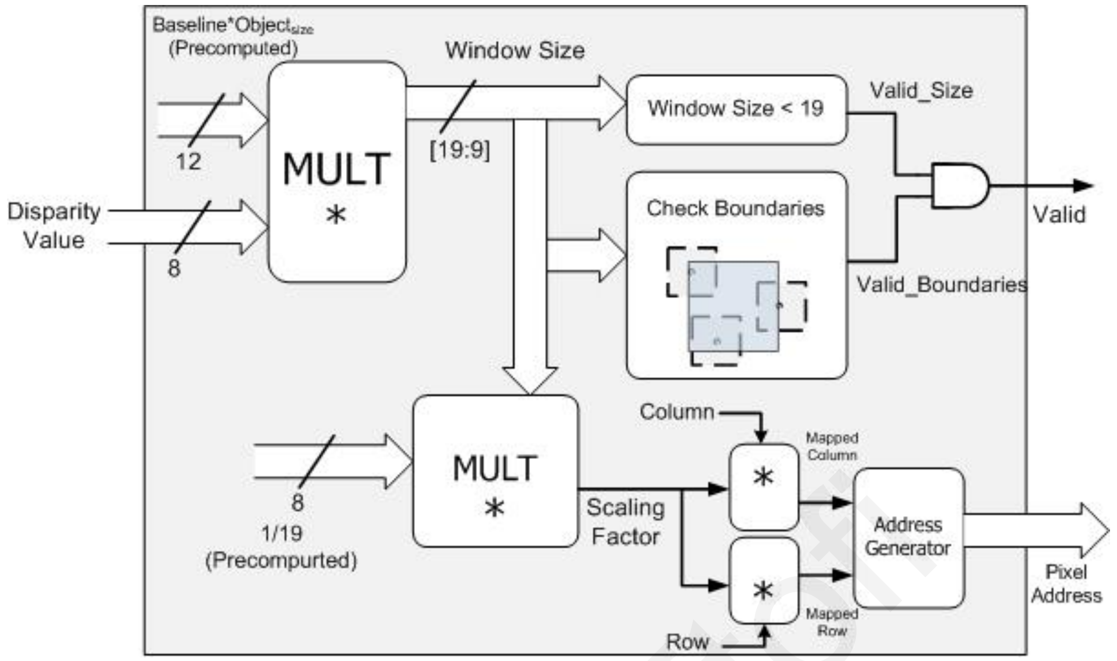


**Figure 6.3: Window Extraction Unit.**

stage usually receives fixed-size search windows (*m x m* in our case), therefore the WEU is responsible for compensating for the different sizes. The WEU performs comparisons to check if the disparity-directed search window is smaller than the classifier window size (*m x m*), and whether it exceeds the original input image boundaries. If one of the conditions is true, the disparity-directed search window is discarded.

In the likely case that the window size is greater than *m x m* but smaller than the original image boundaries, we need to downscale the window to *m x m* so that it fits in the classification stage. We employ the downscaling method as it does not increase the memory and computation demands, even if it can lead to some data loss. Furthermore, in this case we do not need to read the entire disparity-directed search window first and then downscaling it, but rather read only the pixels that are required to form the *m x m* search window from the larger window. This ensures that for every window, regardless of the estimated window size, only $m^2$ pixel values will be read and sent to the CE. To achieve this, we map every coordinate from a *m x m* window onto one coordinate of the larger window, by essentially upscaling the *m x m* window and fetching the pixels from the mapped coordinates. Details of this scaling technique are given in Figure 6.4.

### 6.1.4.3 *Classification Engine*

The classification engine (CE) used in the proposed detection system features a simple, yet powerful Support Vector Machine [**159**] hardware architecture proposed in [**155**]. The

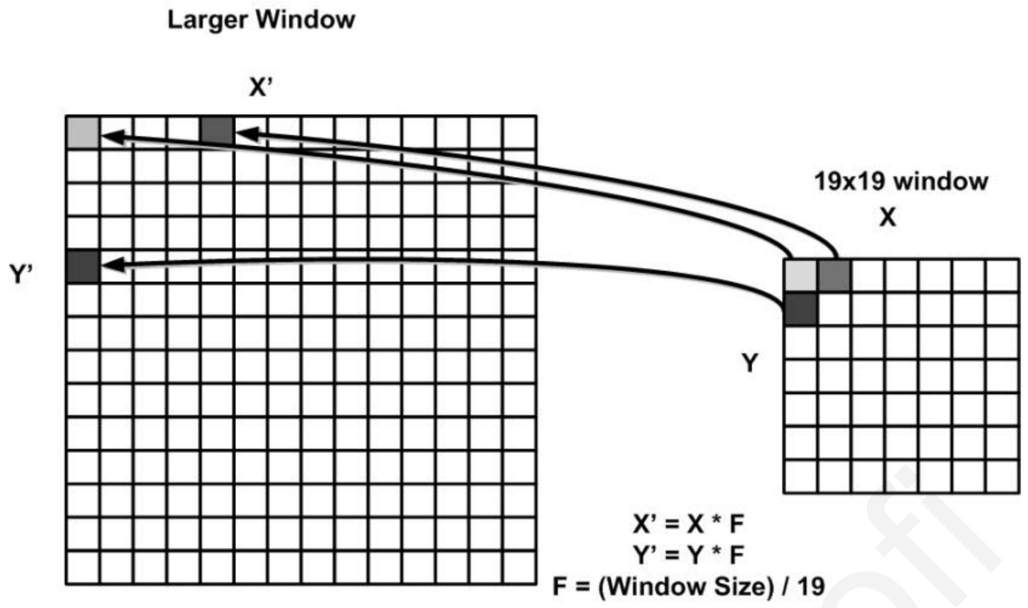


**Figure 6.4: Downscaling Process. Each coordinate in the 19x19 window is mapped to a coordinate in the larger window. The coordinates where the 19x19 window coordinates are mapped, correspond to the pixels values that will be read for classification.**

processing elements are responsible for computing dot-product operations between vectors while dedicated units handle the scalar processing of the SVM computation flow. The classifier is modular and simple; hence, a number of parallel classification units can be used, to account for different performance demands. The classifier receives the *m x m* search window as input, and returns whether the window contains the object of interest or not. The CE architecture and the interconnectivity between processing elements are illustrated in Figure 6.5.

### 6.1.4.4 *Flow of Data*

The overall operation is partitioned into three stages: disparity computation, search window estimation and window classification. The memory controller fetches the stereo image pair from the external memory to the DEU in raster-scan fashion, and stores the incoming pixels in the *input image buffer*. The DEU utilizes these values from the buffer and computes the disparity map, which in turn stores in the *disparity buffer*. Then the WEU can begin the window estimation procedure for the computed disparity values. If the generated window is valid the WEU generates the addresses from which to fetch pixels for that window. The memory controller receives the addresses and starts fetching the pixel data from the *input image buffer*. The incoming pixels are fed to the CE which determines whether the object of interest is present in the window or not.

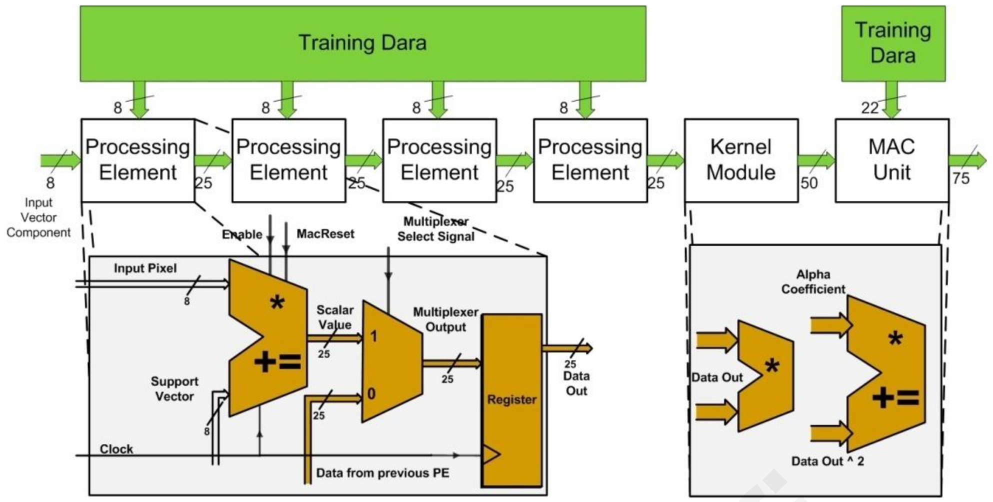


**Figure 6.5: SVM classification engine architecture.**

## 6.1.5 Experimental Platform and Results

### 6.1.5.1 *Experimental Platform and Methodology*

To evaluate the proposed architecture we implemented the system architecture shown in Figure 6.2 using a Xilinx ML505 board (Virtex 5-LX110T FPGA), targeting face detection, a popular object detection application. The Microblaze soft processor [**122**] was used as the system I/O controller to handle tasks such as memory transfers and external I/O. We used a custom built stereo vision system to construct a dataset consisting of ten 240x320 stereo image pairs to evaluate the architecture. The stereo system consists of two video cameras separated by a baseline distance of 77mm, both with a focal length of 25mm. The stereo system was calibrated using the Camera Calibration Toolbox for MATLAB [**47**], and was then used to capture stereo image pairs, which were rectified and stored on the on-board DRAM, and used as input data to the system. Visual output was directed to a digital monitor, through the on-board DVI output Figure 6.6 (b and c) shows the output of our experimental framework.

The training of the Support Vector Machine classifier was carried out in MATLAB using the methodology and strategies employed in [**160**], [**161**]. We used the face detection database from [**162**], which we also enhanced using bootstrapping. The training procedure produced 400 support vectors for a 2nd degree polynomial kernel $(x \cdot y + 1)^2$ [**159**], and all training data were stored on the FPGA Block RAM.

To evaluate the performance of the proposed architecture we setup two different configurations: one using the disparity map to guide the search window space, and one using the traditional sliding window approach. The latter consists only of the SVM classifier and a controller which fetches window data from the input image for various scales. The two systems have the same I/O constraints, memory requirements, and training data. We identify the speedup and hardware overheads of the proposed architecture compared to the traditional object detection approach. Additionally, we also provide results when increasing the number of CEs from one to three, to illustrate the impact of inter-window parallelism as an added benefit of the reduction in search data stemming from the disparity-directed computation.

The sliding window approach requires that the input 240x320 is scaled a number of times to account for different object sizes. As such the input 240x320 image is downscaled five times resulting in additional five images (180x240, 90x120, 45x60, 36x48, 19x19). For each image we generate windows with an overlap of 5 pixels resulting in a total of 4,874 generated windows. On the other hand, for the depth-directed approach we sample the disparity map every 5 pixels to achieve a high granularity and also compensate for erroneous disparity values. In the worst case scenario all sampled disparity map windows will be valid windows resulting in a total of 2496 windows. However, practically this will not be the case, as from a number of experiments we carried out the number of valid windows is on average around 1000, and possibly less depending on the number and size of objects inside the input image.

The performance of depth accelerated object detection system is measured by the time necessary to compute the disparity map of the input stereo pair and the time needed by the classification engine (in our case SVM) to classify the generated windows from one of the two stereo images. The total processing time is affected by the performance of the DEU and the CE. When the DEU performance is greater, the speedup will also be improved. On the other hand, when the performance of the CE is greater, the speedup is constraint by the DEU performance. The experimental results over the input stereo pairs of images we used indicate that by using the depth guided object detection the number of generated windows is reduced

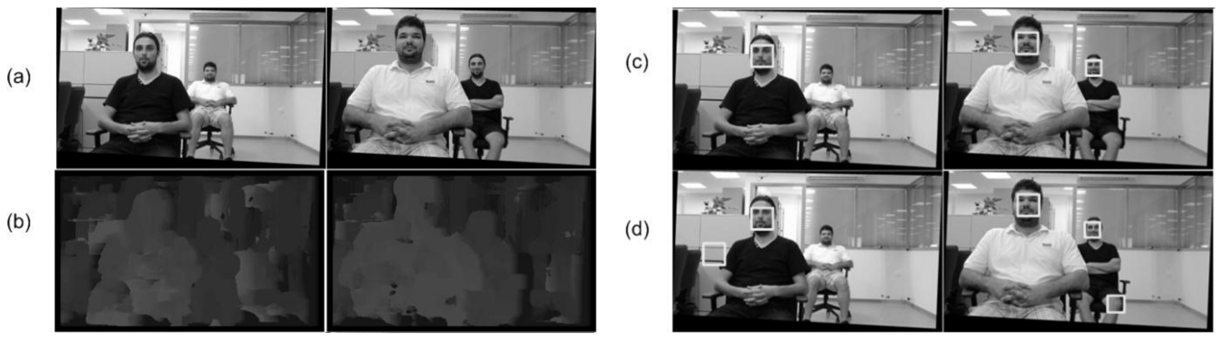


**Figure 6.6: Evaluation Images and Results. (a) Right Images from stereoscopic pairs (b) Disparity Maps from stereo processing (c) Detection results using depth-guided method (d) Detection results using the traditional sliding window approach.**

on an average of 80% (about 3800 less windows) compared to the traditional sliding window approach. As a result, the frame rate is also increased.

This drastic reduction in search windows makes the performance of the chosen classification stage (CE unit) critical in the proposed implementation. However, this is an easily addressable issue, as the classification engine is modular and scalable, thus additional classification engines can be integrated to the system for performance improvement purposes.

#### 6.1.5.2 *Performance Results*

We compare the performance speedups when using the depth-directed search compared to the traditional sliding window search, for a range of 1 to 3 CEs. In all three cases, we observe speedups in the range of 1.9 for the worst case scenario, and 4.7 for the average case. The comparisons are done using the same number of classifiers, with and without the integration of depth acceleration. Three classification engines are needed for real time performance. However, it was observed that when a disparity-directed search is used, only one classifier is necessary for real-time performance (for the average case), yielding both hardware and energy savings. It must be noted that in the current configuration, the bottleneck of the system is the classifier. However, as the number of classifiers increases, the performance bottleneck shifts from the classifier to the disparity computation. Table 6.1 summarizes the performance results for various system configurations for both the average and worst case.

In comparison, the algorithm proposed in [**152**] and implemented in software offers speedup of 2.8, as it does not take advantage of the inherent parallelism of the application. Furthermore, the resulting speedup is also achieved by searching specific regions in the image every two frames instead of one. It must also be noted that [**152**] uses a different classifier, that consumes many resources when implemented in hardware [**147**]. Moreover, the detection

**Table 6.1: Performance for different system configurations.**

| Configuration | | | FPS | # of Windows |
|---|---|---|---|---|
| *# of CE* | *Window Generation Method* | *# of WEU* | | |
| 1 | SW | - | 11 | 4874 |
| 2 | SW | - | 21 | 4874 |
| 3 | SW | - | 32 | 4874 |
| 1 | DA | 1 | 21 | 2496 (worst case) |
| | | | 53 | ~1000(average) |
| 2 | DA | 2 | 42 | 2496 (worst case) |
| | | | 107 | ~1000 (average) |
| 3 | DA | 3 | 64 | 2496 (worst case) |
| | | | 150 | ~1000 (average) |

SW: Sliding Window, DA: Depth Accelerated, CE: Classification Engine results are comparable with those of existing hardware detection systems featuring other classifiers [**148**], [**147**], [**149**], achieving 28-30 frames per second. We observe that in the worst case with one classifier and depth acceleration the performance is comparable with exiting works, while all other scenarios that use depth acceleration outperform the performance of existing works.

Detection accuracies of the SVM CE are similar to other reported works [**160**], [**161**] at about 95% for the given training set [**162**]. While the accuracy of the detector relies heavily on the chosen classifier and the training set and is beyond the scope of this thesis, an inherent advantage of the proposed depth-directed approach also includes a reduction in the number of false-positive detections, as expected. Evidenced by Figure 6.6 (c) and Figure 6.6 (d), when utilizing depth information, the number of samples is reduced and as a result the number of false positive detections is reduced implicitly, improving the system accuracy.

#### 6.1.5.3 *Energy Savings and Hardware Overheads*

We used the Xilinx X-Power Analyzer tool and the input stereo data as input to the system in order to determine the relative energy savings when comparing the sliding window approach vs. the depth-directed approach. We obtained average dynamic power consumption figures, which we then used to estimate the total energy consumption per frame for each system. Results show that there is approximately 48% energy reduction in energy per frame when using 1 CE. Energy savings are observed also as the number of CEs increases; when increasing the number of CEs from 1 to 3, the reduction in energy is 41%. We observe that the savings are reduced however, indicating that the primary source of energy consumption in the optimized system is the CE unit. Overall, the energy savings are attributed to the large amount

**Table 6.2: Hardware requirements for each unit and different system configurations.**

| System | FPGA Resources | | | |
|---|---|---|---|---|
| | ***Slice LUTs (69,120)*** | ***Slice Registers (69,120)*** | ***DSP48E (64)*** | ***Block Ram (148)*** |
| ***WEU*** | 419 (~1%) | 33 (~1%) | - | - |
| ***CE*** | 14,385 (20%) | 5,197 (7.5%) | 8 (%) | 100 (67%) |
| ***DEU*** | 11,996 (17%) | 16,145 (23%) | - | - |
| ***Microblaze*** | 7,016 (10%) | 8,180 (11%) | 3 (4%) | 30 (20%) |
| ***1 CE, SW*** | 21,401 (32%) | 13,377 (19%) | 8 (12%) | 130 (87%) |
| ***2 CE, SW*** | 35,786 (51%) | 18,574 (26%) | 16 (25%) | 130 (87%) |
| ***3 CE, SW*** | 50,171 (72%) | 23,771 (34%) | 24 (37%) | 130 (87%) |
| ***1 CE, DA*** | 33,816 (48%) | 29,555 (42%) | 8 (12%) | 130 (87%) |
| ***2 CE, DA*** | 48,201 (69%) | 34,752 (50%) | 16 (25%) | 130 (87%) |
| ***3 CE, DA*** | 62,586 (95%) | 39,949 (57%) | 24 (37%) | 130 (87%) |

of data that is eliminated from the classification process, which compensates for the disparity computation overheads.

The relative hardware requirements for each major unit are shown in Table 6.2, along with the total hardware requirements per system configuration (number of classification engines and search method). While the area overhead is not negligible the performance speedups are more than enough to justify these overheads.

### 6.1.6 Conclusion

This section provided insights into how depth information can be exploited to reduce the search space involved in the object detection process, thus enabling the development of high-performance and energy-efficient hardware-based object detection systems. A hardware architecture that integrates the edge-directed stereo matching architecture from Chapter 3 with an SVM-based object detector is also presented in this section. Results showed performance speedups ranging from 1.9 to 4.7, with energy savings of 41% - 48%, relative to traditional, sliding window based object detection systems.

## 6.2 Real-Time Obstacle Avoidance for Mobile Robots

An embedded, real-time, and low power obstacle avoidance system is a critical component towards fully autonomous robots that can be used in safety missions, space exploration, and transportation systems among others. This section presents a holistic platform for evaluation of obstacle avoidance systems and autonomous robots, based on reconfigurable hardware. The platform integrates an obstacle avoidance algorithm that uses depth information extracted from a stereoscopic camera. The platform is optimized and implemented using a low-power FPGA-based hardware architecture, enabling its use in battery-operated environments. The platform is comprised of an ATLYS Spartan-6 FPGA board hosting all relevant algorithms, and a modified FDX Vantage 1/10 electric car platform used for navigation. The evaluation under real world conditions indicates that the platform is capable of real-time obstacle avoidance and navigation, with accuracy (~ 92%) equivalent to software implementations.

The rest of the chapter is organized as follows. Section 6.2.1 introduces the problem of obstacle avoidance in robotics and motivates the need for reconfigurable hardware and real-time stereo vision capability. Section 6.2.2 provides information on related work, algorithms and methods. Section 6.2.3 describes the proposed experimental robotic platform detailing the main components and hardware architecture. Section 6.2.4 presents results from real-world evaluation, and finally, Section 6.1.6 provides concluding remarks.

### 6.2.1 Introduction - Motivation

Robots have evolved into a major part of the modern society and are increasingly being deployed in various emerging applications from indoor surveillance to rescue missions. The majority of these systems however is mostly semi-autonomous and requires some form of human guided operation. A fully autonomous mobile robot has to be able to move throughout its operating environment and at the same time work for an extended period without any human intervention. As such, highly efficient and detailed perception is recognized as one of the most important challenges for autonomous and effective robots. In particular, automatic environment recognizing/sensing is a fundamental scientific issue in mobile robotics since such efficient schemes will significantly increase the capacity of the robots to interact with three-dimensional real world environments. One of the most important tasks in autonomous robotics is the computation of accurate three dimensional maps. Here, high resolution precise 3D data as well as fast and accurate matching algorithms / architectures are required to create consistent scenes, which are necessary to enable exploration and navigation in known and unknown terrains. Real-time 3D computation of the scene in the moving direction of a robot is required to ensure obstacle avoidance, which is an important step towards autonomous navigation. This essential task often relies on depth measurements based on laser, sonar and structure-light sensor in the immediate vicinity of the robot. However, as we discussed Chapter 2.2, these sensors are usually associated with short-range operations and bad performance in the presence of strong radiation and direct sunlight, thus they are mostly tailored to indoor environments.

A current trend towards creating highly efficient and detailed perception subsystems in robotics is the incorporation of automated image analysis and computer vision capabilities. Extracting information from visual input is necessary to increase the perception and understanding that robots have of their surrounding environment. Stereo vision one such computer vision-based technique aimed at inferring depth information from a pair of 2D images, is probably the most suitable solution when it comes to reliable depth estimation, especially for long-range operations in both indoor and outdoor situations and under variable lighting conditions. Thus it can particularly benefit the obstacle avoidance process in robotic systems. Specifically, with stereo vision, a robot is able to not only determine if an object or obstacle is in its immediate path, but also to accurately determine its distance, thus improving both obstacle avoidance as well as navigation [**163**].

Real-world applications for mobile robots are often associated with real-time and low power constraints something that limits the available processing resources. Today, camera systems that can deliver 3D-video with a resolution of more than 20 thousand pixels at a rate of more than 30 frames per second are available. Beside the high data rate, the low weight and small size of those systems make them a very interesting sensor for the mobile robotics platforms. However, even though those cameras provide even more than 30 frames per second the existing CPU-based systems cannot execute the necessary cue-extraction and object recognition algorithms, especially when several cues should be extracted simultaneously, at a rate of more than 20 fps. The main reason for the low rates achieved is the fact that the various cue extraction and object recognition schemes are very CPU intensive tasks; for example it has been reported that robust approaches just for object detection based on stereo processing need the performance equivalent of that triggered by more than a dozen high-end CPUs. Obviously, things are getting even more difficult when also considering the CPU load due to the navigation and possible face recognition tasks.

To enable real-time capability, autonomous robots utilize high-end reconfigurable devices; it has been proved that reconfigurable devices in the form of FPGAs allow for extremely higher performance and power-efficient processing when implementing data manipulation methods such as 3D sensing/matching schemes as well as template and feature-based object recognition algorithms, while they can be reconfigured on real-time. This makes FPGAs increasingly attractive for robotic applications. With FPGAs, it is possible to develop real-time vision algorithms, through dedicated parallel hardware architectures, enabling higher response rates so that obstacles can be detected and avoided even at high speeds [**164**]. Advances in FPGA technology also enable low power consumption, facilitating increased battery life and longer autonomous operation. These factors, along with the emergence of stereo vision algorithms suitable for reconfigurable hardware [**165**] are paving the way for the development of computationally effective, low-power stereo-vision-based FPGA-based mobile robots that can reliably avoid obstacles in unconstraint environments.

There have been several attempts so far towards developing obstacle avoidance and navigation systems on real-world robotic platforms [**166**], [**167**], [**168**]. However, the majority of them employ sonar and laser sensors to perform depth measurements. Only few systems have utilized vision-based techniques, but these rely on software running on a laptop computer [**169**], [**170**], [**171**] for the implementation of stereo vision technology, obstacle avoidance, and/or navigation algorithms. This increases the size and weight of the robotic platform, hence decreasing its battery life. On the other hand, hardware-oriented implementations such as [**164**], consider only part of the computations on the FPGA, and neglect platform implementation issues.

In this thesis a complete prototyping platform for the evaluation of obstacle avoidance systems and autonomous robots is realized on reconfigurable hardware. An efficient stereo vision algorithm for producing the necessary 3D and an obstacle avoidance subsystem were both implemented on an ATLYS Spartan-6 FPGA board equipped with a *VmodCam* stereo camera module. A modified *FDX Vantage 1/10* electric car platform was used for testing the proposed architecture in indoor and outdoor real-world scenes. The system receives stereo image data from the *VmodCam* module and a decision-making algorithm is applied on a specified *Region of Interest* (RoI) on the produced disparity map. The algorithm outputs the direction that the robot should move to in order to avoid any obstacles present. Experimental evaluation results indicate that the FPGA-based robotic platform can avoid obstacles in real-time (i.e. can process and identify obstacles within a 1/30th of a second that a stereo image takes to be processed) in both indoor and outdoor environments, with 91.7% accuracy.

### 6.2.2 Obstacle Avoidance - Overview & Existing Techniques

Obstacle avoidance is a classical problem in robotics and is a hot topic in the field of autonomous navigation. It is defined as the task of detecting objects or people in a projected path and steering around them, while avoiding other objects or hazards. The presence of unexpected obstacles in the route of a mobile robot is a real possibility, especially when navigating in a dynamic or unknown environment. Incorporating an obstacle avoidance system is critical in ensuring the safety of the robot as well as its surroundings. Such a system is responsible for detecting the presence of an obstacle and also decides the direction that the mobile robot needs to follow while maintaining the path towards its targeted (original) destination.

Various methods have been proposed for detecting and avoiding obstacles found in the path of a robot agent. Some early methods propose the use of ultrasonic sensors, in order to detect the edges of an obstacle and avoid it [**166**], [**172**]. Others make use of an evenly spaced grid and calculate the probability of occupancy for each cell [**167**], [**168**]. The elastic strips method [**173**], [**174**] treats the trajectory of the robot as an elastic material to avoid obstacles.

The sensor systems used for implementing most of the obstacle avoidance algorithms can be divided in two categories. The first category includes all the systems that are based on laser and IR sensors for providing depth information of their surrounding environment.The emergence of sensors such as Kinect [**36**] and LeapMotion [**175**] enabled the development of accurate and easy to implement autonomous robots [**169**], [**176**]. However the poor performance of such sensors in pure daylight makes them unsuitable to be used in an outdoor environment. The second category includes vision based systems. In [**177**] a vision-based system that uses a single camera and ultrasonic sensing is proposed. It requires a high amount of sensors however. Stereo vision systems on the other hand, are widely used, but most of them require complex computations, therefore require high-end resources to enable real-time processing when implemented in software, making them unsuitable for use on an autonomous robot with limited hardware/power resources.

Recently, stereo vision algorithms suited for reconfigurable hardware have been proposed [**165**]. There exist several stereo vision systems today based on dedicated hardware implementations [**86**], therefore the incorporation of depth information for several vision-based applications in embedded environments is becoming a reality. Nalpantides et al. in [**178**], [**170**] present a simple algorithm for obstacle avoidance with the sole use of a stereoscopic camera. Both stereo vision and obstacle avoidance are implemented in software running on a laptop computer. Thus, this system is not suitable for use in resource-limited embedded environments. However, the obstacle avoidance algorithm presented in [**178**] is quite simple, and can easily be adopted for an FPGA implementation, preventing the obstacle avoidance system from becoming a significant drain on the battery life of the robot.

A stereo vision-based obstacle detection system is presented in [**164**]. It uses the Sum-of-Absolute-Difference (SAD) algorithm for disparity map calculation and proposes the use of region of interest to limit the region of the images over which the disparity map is estimated, in order to speed up the overall process. However, this implementation utilizes a naive approach that only detects the obstacle, while no decision-making is incorporated. Moreover, this is not an embedded solution, since it uses MATLAB to capture the stereo images, where only the SAD calculations are performed by the FPGA. As a result, this system provides only near real-time processing speed (around 20 fps).

Motivated by the advances in recent works, the contributions of this work focus on the development of a holistic prototyping platform integrating an obstacle avoidance algorithm that uses depth information extracted by a single stereoscopic camera, and is entirely based on reconfigurable hardware (FPGAs) optimized for low-power and portable and flexible integration within a mobile FDX Vantage 1/10 electric car platform.

### 6.2.3 Proposed Robotic Platform

#### 6.2.3.1 *System Overview*

The proposed autonomous robotic platform features a stereo camera mounted on a standard electric car that carries an FPGA board, which performs all the processing on board and in real-time. The car is able to navigate around avoiding obstacles with the aid of stereo vision technology, and chooses as the best way to follow, the path that is less likely to contain nearby objects. The proposed prototyping platform is comprised into different subsystems as shown in Figure 6.7. The system consists of an Atlys FPGA board [**179**] that is interfaced to a *VmodCAM* Stereo Camera Module through a VHDCI cable. The FPGA is the main processing unit of the system that implements the stereo vision and obstacle avoidance algorithms, through which the direction that the robot needs to follow is extracted. The decision is then forwarded to a *Raspberry Pi* computer [**180**] through a Universal Serial Bus (USB) interface. The *Raspberry Pi* is connected to an *Arduino* computing platform [**181**] that generates the analogue signals used to steer the robot to the appropriate direction. These main components comprise a platform that is computationally efficient and low-power and can reliably avoid obstacles in unconstraint indoor and outdoor environments. Each subsystem as well as the complete platform is analyzed in the following subsections.

#### 6.2.3.2 *Stereo Vision Subsystem and Disparity Computation*

The stereo vision subsystem implemented on the FPGA receives two independent simultaneous image feeds from the *VMODCAM* Stereo Camera Module and generates a disparity map, from which information about the depth of objects in the image frame relative to the position of the stereo camera module can be extracted. The stereo subsystem consists of three major stages: Cost Volume Construction (CVC), Cost Volume Filtering (CVF) and Disparity Selection (DS). The CVC stage calculates a matching cost for each pixel $p$ at all possible disparities. The output is a three-dimensional structure consisting of $D$ cost images (Stereo Cost Volume - SCV). Each cost is computed as the truncated absolute difference of colors and gradients, a metric that exhibits good robustness to illumination changes [**142**]. The

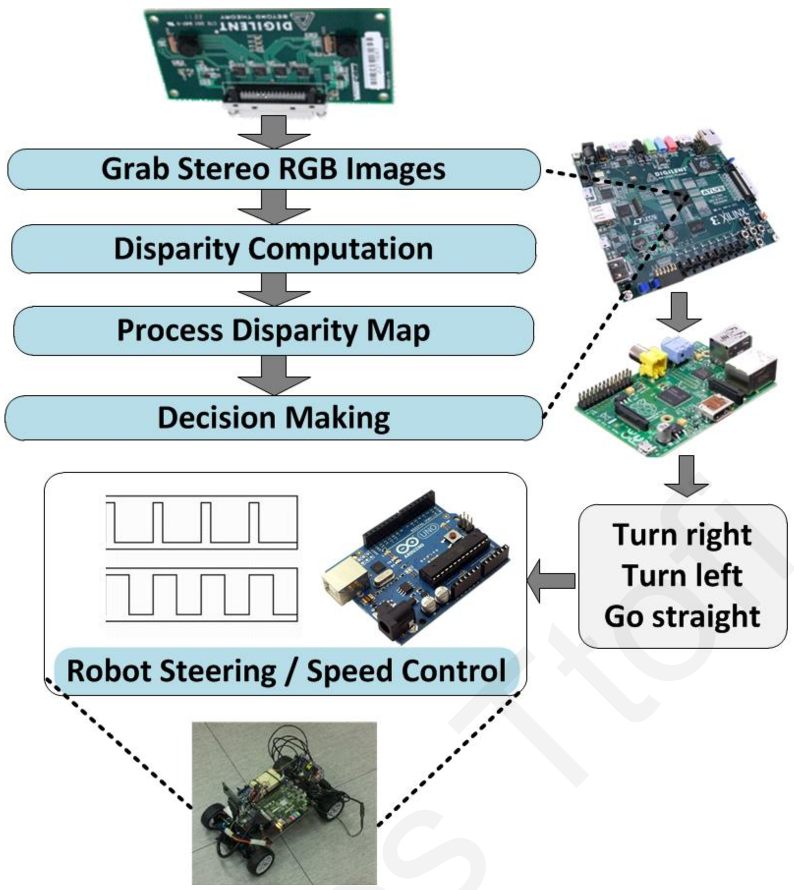


**Figure 6.7: Block diagram of the proposed robotic platform.**

overall cost function *C(p,d)* is computed using (6.3)-(6.5), where *a* is used to balance the influence of the color and gradients terms, and $T_c$ and $T_g$ are truncation thresholds. Truncation is important in order to suppress the influence of noise in the final disparity maps, especially for the indented application which aims to be applied to outdoor scenes that usually suffer from noise caused by lighting differences and reflections.

$$M(p,d) = \sum_{i=1}^{3} \left| I_{left}^{i}(p) - I_{right}^{i}(p-d) \right| \tag{6.3}$$

$$G(p,d) = \left| \nabla_x \left( I_{left}(p) \right) - \nabla_x \left( I_{right}(p-d) \right) \right| \tag{6.4}$$

$$C(p,d) = a \cdot min\left(T_c, M(p,d)\right) + (1-a) \cdot min\left(T_g, G(p,d)\right) \tag{6.5}$$

The CVF step smoothes each slice of the SCV using a typical mean filter, where the filtered output value at *p* and disparity *d* is the sum of pixels in the same slice of the SCV divided by the total number of pixels in the match window as shown in (6.6).

$$q(p,d) = \frac{1}{N} \sum C(p,d), \quad \text{N: total pixels in the window} \tag{6.6}$$

In contrast to most of the existing stereo vision hardware implementations, which compute the sum of pixels in a window [**86**] in a naive approach, the stereo vision system implemented in this work utilizes the moving sum technique to realize an efficient mean filtering process where the resources required are independent of the match window size. In this way, the CVF stage is kept compact consuming less hardware resources, which is important to keep power consumption low and also to be implemented on smaller FPGA devices.

The main idea of the moving sum technique is to maintain a sum for each column in the image to be filtered. Each column sum accumulates *2r+1* pixels, while the window sum is computed by adding *2r+1* adjacent column sums. While filtering the image, the column sum to the right of the window is yet to be computed for the current row, so it is centered one row above. Therefore, the first step consists of updating the column sum to the right of the window, by subtracting its topmost old pixel and adding one new pixel below it. The second step moves the window to the right and updates the window sum by subtracting its leftmost column sum, and adding the updated column sum computed in the first step. The hardware design of the moving sum technique with all the associated design choices and tradeoffs are described in Chapter 5.3.

Once all slice of the SCV are filtered, the DS stage selects the best disparity for pixel p through a simple Winner-Takes-All minimization approach (6.7).

$$d_p = \underset{d \in D}{\operatorname{argmin}}\, q(p,d) \tag{6.7}$$

### 6.2.3.3 *Obstacle Avoidance Subsystem*

The proposed obstacle avoidance subsystem is responsible for detecting and avoiding any obstacles present in the robots' path, to facilitate autonomous navigation. Obstacle avoidance is performed using the previously calculated disparity map frame. Since the robot is moving on a ground level, a region of interest (RoI) is selected and the lower third of the image is ignored in the calculations. The selected RoI is equally divided in three zones, as shown in Figure 6.8 (b).The decision is based only on summations, which are used as an input to a simple FSM responsible for indicating the *safest* direction to follow. In this manner, the safest direction is determined as the one corresponding to the zone containing objects which are far away from the stereo camera module.

For each of the three zones, a pixel based analysis is performed in order to determine how many pixels have a disparity value, *D(p)*, greater than a predefined threshold, *T*. The *T* threshold is empirically defined and it selected in regard to which distance is considered close enough to the mobile agent. The accumulated values are stored in three different registers, *Rl*, *Rc*, *Rr*. where:

$$Ri = \sum_{region\ i} (D(p) > T) \tag{6.8}$$

The final decision is taken by a MOORE State Machine, with three different states. *State 0* outputs an instruction for moving forward, *state 1* for steering left and *state 2* for steering right. The FSM is in state 0 as long as *Rc* is smaller than a predefined rate *r* of the total number of pixels in the window. This means that there is enough space for the robot to move forward. If this is not the case the next state is selected by checking which of the registers *Rl*, *Rr* is smaller. After a steering command is given, the FSM returns to *state 0*. The output of the FSM is saved in the instruction register, *Ri*. An overview of the overall algorithm is shown in Figure 6.9.

The proposed architecture (Figure 6.10) for the decision-making module consists of a decoder, three accumulators, an FSM, and the four registers. The disparity value *D(p)*, and the pixel coordinates in the form of image rows and columns, are given as input from the

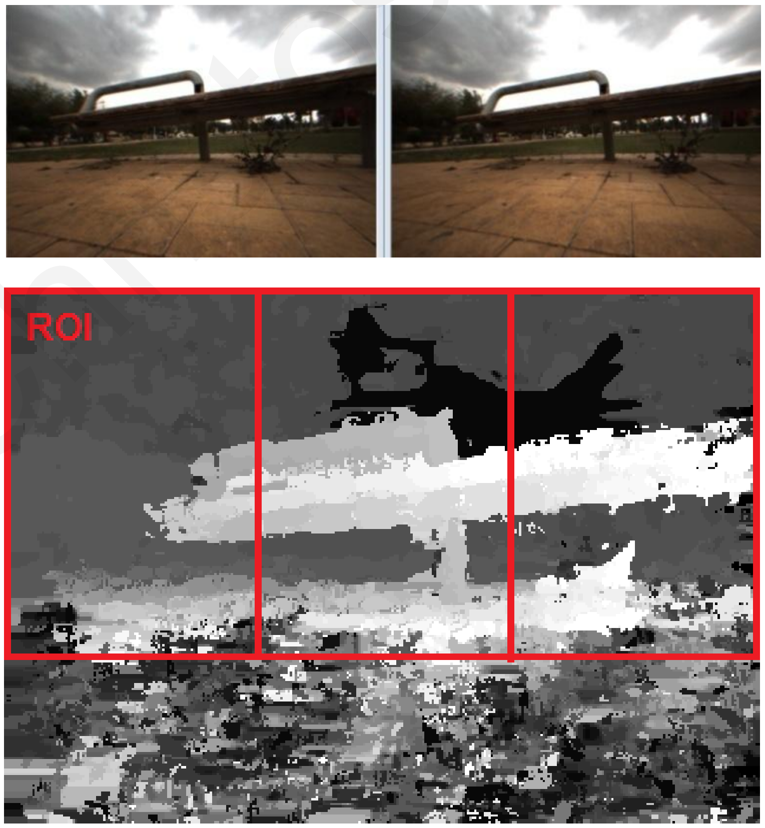


**Figure 6.8: Obstacle Avoidance Algorithm. (a) Input from Stereo Camera. (b) The region of interest divided in three equally sized windows over the disparity map.**

***Next State Process:***

```
1.      next_state<= state - - default is to stay in current state
2.      case (state) is
3.      when state0_go_straight =>
4.          if (Rc< r) then
5.          next_state<= state0_go_straight;
6.          elsif (Rl<Rr) then
7.          next_state<= state1_steer_left;
8.          Else
9.          next_state<= state2_steer_right;
10.     when others =>
11.         next_state<= state0_go_straight;
12.     end case;
```

***Output Process:***

```
1.  if (state = state_go_straight) then
2.          Ri = 00;  - - go straight
3.  elsif (state = state_steer_left) then
4.          Ri = 01;  - - steer left
5.  Else
6.  Ri=10;              --steer right
```

**Figure 6.9: The Next State and Output Process of the Decision-Making Module.**

Disparity Unit. A decoder is used first to define if the pixel is in the region of interest and secondly in which of the three regions the pixel belongs. The disparity value is discarded if it does not belong to the RoI, otherwise it is passed to the respective accumulator. A change frame signal also comes from the Disparity Unit which works as the reset signal for the registers *Rl*, *Rc*, *Rr* and as a write enable for the instruction register *Ri*. The value of the instruction register is then passed in a FIFO structure to facilitate the communication protocol described in the following section.

### 6.2.3.4 *Communication Protocol with the Mobile Agent*

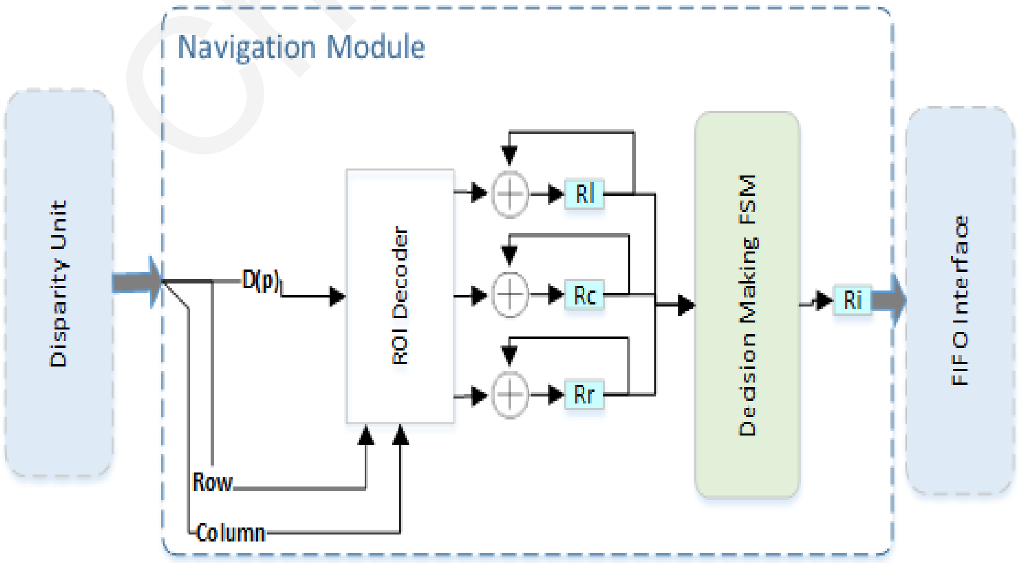


**Figure 6.10: Proposed navigation module architecture.**

In order to transfer the instruction to the robot, a flexible and low-power communication protocol was established by connecting the FPGA board via a USB port with a *Raspberry Pi* (RPi) ARM-based computer that is cheap, portable, and extremely flexible. The RPi has a Broadcom BCM2835 system on a chip (SoC), which includes an ARM 700MHz processor and runs Raspbian OS. In addition to the reasons stated earlier, the particular computer was selected also because of its variety of I/O ports. It includes two USB ports, one of which was used for data communication with the FPGA board, and the other one for powering up the FPGA board. Furthermore its additional GPIO pins allow it to drive the microcontroller responsible for the steering.

The communication between the FPGA and the RPi was established using the FPGALink library [**182**]. The library allows the programming of the board, but at the same time provides a good medium for the host (RPi) to exchange data with the FPGA, once programmed. On the host side a simple *C* routine was used, while on the FPGA side a standard FIFO interface was implemented. The routine running on the host side was responsible to read the instruction Register from the FIFO structure on the FPGA side. The instruction was then passed to the mobile agent using the GPIO pins on the RPi.

### 6.2.3.5 *Mobile Agent and Steering*

To test the system implemented we used a modified FTX Vantage 1/10 remote controlled car, equipped with a 3215kV brushless motor and a servo motor. Both the speed and the steering angle of the car were controlled with pulse width modulation (PWM) pulses. The frequency of the PWM pulses is 50Hz and the amplitude 0V-5V. The information is encoded in the width of the pulse as Figure 6.11 demonstrates. A 25% duty cycle is used for steering

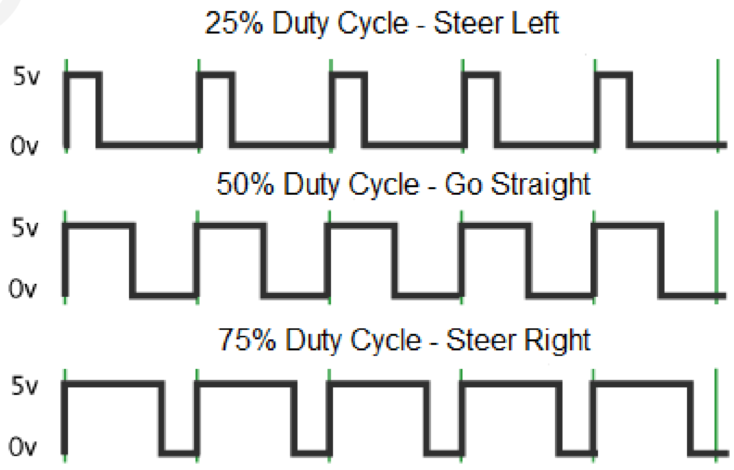

25% Duty Cycle - Steer Left
5v
0v
50% Duty Cycle - Go Straight
5v
0v
75% Duty Cycle - Steer Right
5v
0v

**Figure 6.11: Steering Control with PWM Pulses.**

left, a 50% duty cycle is used for going forward and a 75% duty cycle is used for steering right. As for the speed a 50% duty cycle is used when moving forward and a 25% duty cycle is used when steering.

The receiver of the RC car was bypassed, and an Arduino Uno R3 physical computing platform was used for the control. The Arduino is equipped with an Atmel AVR ATmega328 microcontroller, which receives the instruction from the RPi and generates the corresponding PWM pulses. If an instruction for moving forward is received the steering angle is kept at 0° and the speed remains constant. If a turn left or turn right instruction is received the speed is reduced and the steering angle changes to -30° or +30° accordingly, in order to avoid the obstacle.

#### 6.2.3.6 *Powering the Experimental Platform*

In order to maintain a small physical size and low weight it was important to power the whole platform using only a single power source. To achieve this,a single 3000mAh 7.2V NiMH battery was directly connected to the car and the Arduino. The Arduino was equipped with a voltage regulator and has a 5V output pin, which was used for powering the RPi. The FPGA board was in turn powered with a modified USB cable connected to the RPi.

### 6.2.4 Experiments and Evaluation

#### 6.2.4.1 *Methodology and Results*

To evaluate the performance of the proposed system a series of tests were ran in indoor environments with the lights on and outdoor environments with natural lighting. The robot was placed in front of various scenes to check if it will move towards the right direction by avoiding any obstacles present.

Threshold *T* was 0.6 (for $0 < D(p) < 1$) and the rate *r* was 40% of the total pixels in the central window, for indoor environments and 20% for outdoor environments. These values define the sensitivity of the system and were empirically chosen in a way that the system would navigate safely but at the same time would not be susceptible to false alarms. The reason for choosing different *r* in the case of outdoor testing is because under natural lighting the disparity maps produced have different values. An adaptive rate *r* can be chosen by analyzing image and filtering the image prior to the disparity computations but is left as future improvement as it is not the main focus of this work.

A sequence of 23 test set image pairs (640×480 resolution) were used in closed environments (Figure 6.12) with various obstacles, such as persons, chairs, walls and doors. In 20 out of the 23 cases the robot moved towards the right direction. The results are summarized in Table 6.3.The cases when a wrong direction was selected are highlighted. In all of these cases the system correctly decided to steer in order to avoid an obstacle, but the wrong direction was picked, probably due to noise in the disparity map and poor lighting conditions. Another sequence of 13 test set image pairs (640×480 resolution) was used in outdoor environments (Figure 6.12), with obstacles such as trees and benches. The robot selected the correct direction in all of the cases, which are shown in Figure 6.12. An overall accuracy of 91.7% was achieved during the above tests. Compared to the software implementation presented in [**178**], the accuracy achieved is practically the same, proving that a hardware implementation will not affect the performance of the system.

Another important aspect of the proposed system it's the certainty (*cert* in (6.9)*)* of the

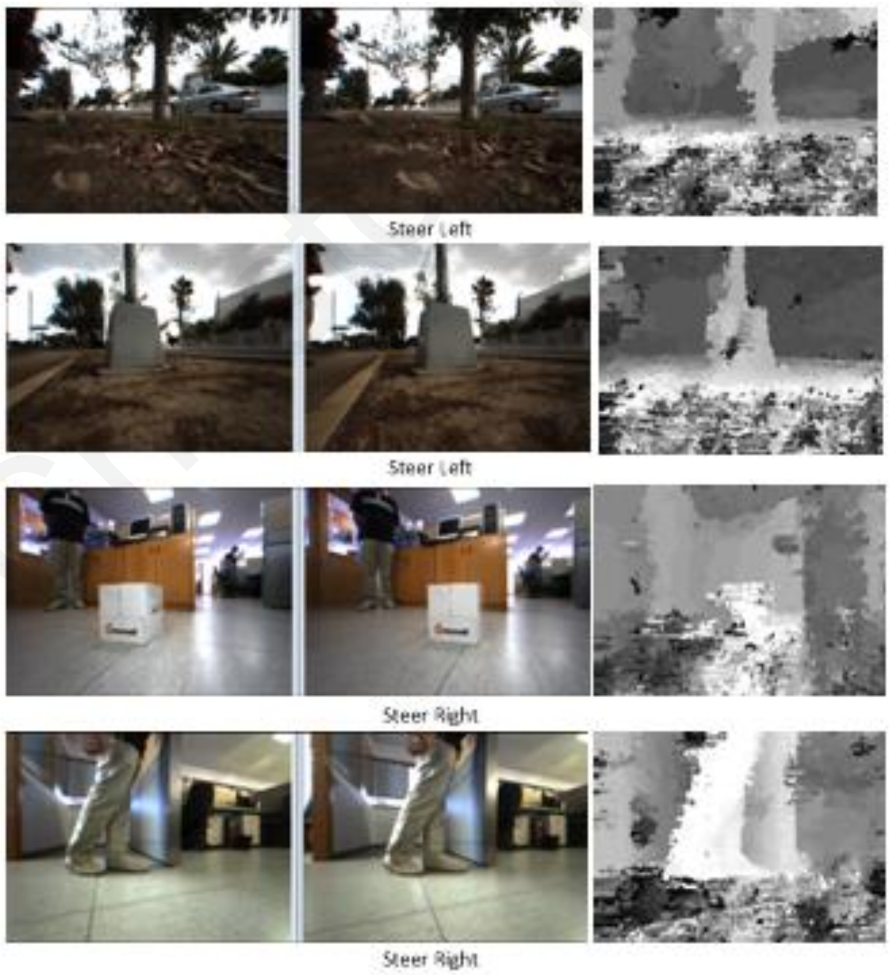


**Figure 6.12: (Top) Two indoor example images. (Bottom) Two outdoor example images with the decision beneath.**

Table 6.3: Evaluation results in indoor environments.

| Test Set | Rl | Rc | Rr | % of central windows with D(p) > T | Cert | Direction Followed |
|---|---|---|---|---|---|---|
| **1** | 5215 | 15305 | 4378 | 93.15% | 16% | right |
| **2** | 5588 | 13088 | 6902 | 80.70% | 19% | left |
| **3** | 9048 | 3713 | 10506 | 22.74% | - | straight |
| **4** | 8752 | 11555 | 9755 | 70.32% | 10% | right |
| **5** | 5360 | 14652 | 2102 | 89.75% | 61% | right |
| **6** | 9822 | 2511 | 2071 | 15.38% | - | straight |
| **7** | 6794 | 11081 | 10735 | 68.32% | 37% | left |
| **8** | 4048 | 9856 | 13648 | 60.77% | 70% | left |
| **9** | 5618 | 3808 | 2154 | 23.48% | - | straight |
| **10** | 4335 | 479 | 2470 | 2.93% | - | straight |
| **11** | 9393 | 11029 | 5869 | 67.56% | 38% | right |
| **12** | 9145 | 7453 | 7350 | 45.95% | 20% | right |
| **13** | 3826 | 3088 | 1153 | 18.91% | - | straight |
| **14** | 412 | 7511 | 3215 | 46.01% | 88% | left |
| **15** | 3417 | 12990 | 12623 | 79.57% | 72% | left |
| **16** | 6169 | 7687 | 4372 | 47.09% | 30% | right |
| **17** | 4204 | 2668 | 7806 | 16.45% | - | straight |
| **18** | 5220 | 7380 | 9554 | 45.50% | 46% | left |
| **19** | 7000 | 7124 | 5208 | 43.92% | 26% | right |
| **20** | 8124 | 3267 | 4701 | 20.01% | - | straight |
| **21** | 3108 | 3458 | 8353 | 21.18% | - | straight |
| **22** | 4123 | 8983 | 8572 | 55.38% | 52% | left |
| **23** | 10802 | 4704 | 9550 | 28.81% | - | straight |

decision-making module, when a left or right steering direction needs to be chosen. This is calculated with a comparison between the *Rl*and the *Rr* registers and is defined as in (6.9). The results obtained are at the same levels of the software implementation in [**178**], and in some cases are much higher. When both the left and the right direction are traversable the certainty of the decision is relatively small. However the obstacle avoidance task is still fulfilled.

$$cert = \frac{Rmax - Rmin}{Rmax} \tag{6.9}$$

### 6.2.4.2 *Hardware Overheads and Power Consumption*

The total hardware requirements of the FPGA board are shown in Table 6.5. A processing performance of 30 frames per second was achieved, which implies that the developed platform can avoid obstacles in real time situations. The overall power consumption of the proposed architecture, excluding the electronic car platform, is 6 W, making it ideal for use on mobile robots, without becoming a significant drain on its battery life.

Table 6.4: Evaluation results in outdoor environments.

| Test Set | Rl | Rc | Rr | % of central window with D(p) > T | Cert | Direction Followed |
|---|---|---|---|---|---|---|
| **1** | 3551 | 6111 | 2432 | 37.19% | 32% | right |
| **2** | 2821 | 8176 | 2021 | 49.76% | 28% | right |
| **3** | 1501 | 4880 | 2446 | 29.70% | 37% | left |
| **4** | 2451 | 2938 | 2988 | 17.88% | - | straight |
| **5** | 2145 | 5793 | 3676 | 35.25% | 42% | left |
| **6** | 2616 | 3571 | 2778 | 21.73% | 5% | left |
| **7** | 1073 | 1908 | 1665 | 11.61% | - | straight |
| **8** | 2004 | 6538 | 2235 | 39.79% | 10% | left |
| **9** | 1324 | 2089 | 1707 | 12.71% | - | straight |
| **10** | 3626 | 8153 | 1662 | 49.62% | 54% | right |
| **11** | 213 | 1256 | 432 | 7.64% | - | straight |
| **12** | 439 | 1906 | 2368 | 11.60% | - | straight |
| **13** | 2467 | 7079 | 6184 | 43.08% | 60% | left |

Table 6.5: FPGA Platform Parameters.

| | | | | | |
|---|---|---|---|---|---|
| **FPGA Platform** | Spartan 6 Digilent XC6SLX45-CSG324 | **FPGA LUTs** | 13576 (50%) | **Frequency (MHz)** | 124 |
| **FPGA BRAM** | 56 (55%) | **FPGA Registers** | 7402 (13%) | **Frame-Rate** | 30 fps |

### 6.2.5 Conclusion

This chapter presented a real-time and low power stereo vision-based mobile obstacle avoidance platform that can be used for prototyping and experimental purposes. The stereo vision and obstacle avoidance algorithms were both optimized and implemented with an FPGA-based hardware architecture, and were utilized to develop a complete experimental platform based on a modified FDX Vantage 1/10 electronic car platform. Evaluation results obtained under real-world conditions in both indoor and outdoor environments indicate that the proposed platform is capable of real-time, low-power obstacle avoidance and navigation, with accuracy equivalent to software implementations.

# CHAPTER 7

# Conclusion and Future Work

*THIS thesis investigated hardware architectures of stereo matching algorithms that have the potential to satisfy the requirements of constrained embedded vision applications. Initially, a stereo matching architecture that utilizes edge information as a means to accelerate the overall matching process, and reduce its logic and memory requirements, was presented. Afterwards, the thesis introduced hardware design optimizations that can be applied to a complex, accurate matching algorithm that uses image segmentation and adaptive support weights (ADSW), in order to obtain an effective speed-accuracy tradeoff. Moreover, the thesis discussed how the properties of the recently proposed Guided Image Filter, and particularly its edge-preserving property and linear-time complexity, can be exploited to reduce the hardware complexity of the ADSW cost aggregation process, while maintaining high quality results, and also to improve the effectiveness of the disparity refinement step through weighted median filtering with the guided filter weights. Finally, the thesis provided insights obtained from evaluating the proposed architectures in object detection and obstacle avoidance applications. This chapter summarizes the keys points of the work done in the thesis and discusses directives for future research.*

## 7.1 Concluding Remarks

Stereo vision, the task of matching the images taken from a stereo camera and extracting the depth of objects in a scene, is widely used in several embedded applications such as intelligence surveillance, autonomous vehicles and mobile robots. However, stereo vision is a computationally demanding task, and thus it is a major challenge to produce 3D content in real time with high quality and under limited power and hardware resources. Traditional CPUs fail to take advantage of the inherit parallelism of stereo vision algorithms and thus struggle to achieve real-time frame rates. On the other hand, GPUs have high power consumption demands. FPGAs seem to offer a satisfactory trade-off between performance, power, flexibility and programmability. Thus, this research has focused on the implementation of three different stereo matching hardware architectures, each satisfying specific constraints in embedded environments.

The thesis first presented a stereo matching hardware architecture that incorporates an edge detection mechanism to direct the matching process only on important features on the stereo images, and therefore reduce the overall data that needs to be processed. The proposed edge-directed architecture targets resource constrained embedded systems with hard real-time requirements. However, the architecture relied on a local stereo algorithm with fixed window aggregation, while many of today’s applications demand the extremely high matching accuracy offered by more complex stereo algorithms.

Hence, the thesis also presented the design of a segmentation-driven stereo algorithm that uses adaptive cost aggregation, in an attempt to increase the robustness of the matching process. It also presented hardware-oriented algorithmic modifications and optimization techniques that make the algorithm hardware-friendly and suitable for efficient dedicated hardware implementation. The segment-based stereo matching architecture provides an effective balance between speed and accuracy aspects in embedded vision systems, however at the expense of high resource usage.

Therefore, the research related to this thesis has also focused on the design of a fully pipelined, parallel and scalable stereo matching hardware architecture based on the recently proposed Guided Image Filter. FPGA simulation results extracted by integrating this type of filter in the cost aggregation step of ADSW-based hardware stereo matching systems indicate that the overall hardware complexity of cost aggregation is reduced significantly, which in turn could allow for real-time stereo matching of HD images, as well as improvements of the overall matching accuracy, thanks to the edge-preserving property of the GIF. In addition edge-preserving nature of the GIF has been exploited in order to design a power disparity refinement architecture, which can be combined even with fixed support cost aggregation to result in resource-efficient, high-quality stereo matching design.

As a whole the research done in this thesis aims at the design of novel and efficient architectures and design techniques that can be used in various embedded vision systems to perform operations that can benefit from the use of depth information. Depending on the targeted application domain and associated constraints, each of the presented architectures might be more suitable for particular embedded vision applications. Additionally, it is possible to explore the real-time reconfiguration aspect of today’s FPGAs to use the developed architectures at any time depending on the actual state of the environment. The thesis has presented an evaluation of the edge-directed architecture on a depth-accelerated hardware object detection system. Future plans include the evaluation of the architectures on autonomous navigation for unmanned vehicles.

## 7.2 Future Directives

### 7.2.1 Short-Term Plans

In this section, we attempt to combine all insights from the thesis to suggest how the proposed depth computation architectures can be improved in a future system design. The proposed improvements are discussed in the following subsections.

#### 7.2.1.1 *Edge-directed stereo matching architecture*

The edge-directed disparity estimation architecture presented in Chapter 3 can be improved in several ways. First, the Sobel edge detector threshold was determined empirically by finding the threshold yielding the lowest matching error in terms of percentage of bad pixels, and computing the average for a number of stereo pairs in the Middlebury dataset. An improvement to this approach would be to extend the implemented edge detection unit to compute the threshold dynamically, either by computing a threshold value for each image (e.g. by considering the image mean value or histogram), or by assigning different thresholds for different regions in the image (e.g. a threshold for local neighborhood). Second, the SAD values between correlation windows were computed using parallel XOR circuits and adder trees. While these operations were performed on binary data based on low-complexity adder trees (in terms of bit-width), the use of the adder trees could be avoid if the cost computation and aggregation steps are interpreted as a smoothing filter of the SVC. In this way, the SAD values can be computed by using the mean filter architecture described in Chapter 5.3, therefore replacing the adder trees with 1-bit block RAMs and simple adders/subtractors. Third, the interpolation mechanism integrated in the edge-directed stereo matching architecture has been implemented on an embedded soft processor core, and works by filling the non-edge points with disparity values from the local neighborhood of the non-edge point being processed, using the nearest neighbor interpolation method. The efficiency of the interpolation process could be improved by implementing and integrating custom hardware units of more complex interpolation methods such as bilinear and bicubic interpolation. Finally, the edge-directed architecture does not integrate a disparity refinement pipeline. As such, there is still room for improving the disparity results by integrating post-processing steps, like the ones integrated by the other two stereo matching architectures presented in this thesis. Especially, the GIF-based weighted median filtering presented in Chapter 5 has the potential to increase the matching quality significantly.

#### 7.2.1.2 *Segment-based ADSW stereo matching architecture*

The segment-based ADSW architecture (Chapter 4) integrated a simple segmentation method based on thresholding of the pixel values. We expect that a more advance segmentation method (e.g. k-means segmentation) could improve the confidence of the generated adaptive weights. Moreover, the current version of the architecture uses sub-sampled versions of the correlation windows in order to reduce the cost aggregation's complexity. Although this approach reduces the hardware logic required (due to the use of a smaller number of weight generators and cost aggregators), the working principle of the scanline buffers imposes that the size of the on-chip memory still depends on the width of the original window. An improved approach would be to utilize correlation windows with a cross-based template. In this way, the pixels that contribute to the correlation process are located across a single row and a single column of the entire correlation window. The pixels across the row can be stored in a shift register (its size depends on the correlation window width), while access to the pixels across the columns of the correlation windows can be achieved by utilizing a $Wxm$ block ram ($W$: width of image, $m$:width of correlation window). If configuring the block ram so that each memory location stores $m$ pixels, then a single access to the block ram can provide parallel access to the pixels across a column of the correlation window. This memory location then needs to be updated by shifting the memory content to the right and concatenate the new pixel.

#### 7.2.1.3 *Depth-guided embedded vision applications*

The depth-directed object detection application presented in Chapter 6.1 can be evaluated using alternative classification engines (e.g. AdaBoost-based classifier or Neural networks), in order to further explore the impact of the different algorithm-specific parameters, and investigate the benefits of incorporating edge information with other classification engines. Furthermore, we plan on comparing the depth-directed approach with alternative ways of reducing the search spaces such as edge detection., Finally, the incorporation of path planning and navigation into the stereo vision-based mobile obstacle avoidance platform presented in Chapter 6.2 could make the platform an ideal test bench for fully autonomous navigation.

### 7.2.2 Long-Term Plans

The work started by this dissertation can be further continued in many directions, which are summarized in the following subsections.

#### 7.2.2.1 *Development of a Stereo Vision Based Adaptive Computing System*

This thesis developed different real-time stereo vision hardware architectures, optimized for the specific requirements of embedded environments. The edge-directed stereo vision architecture targets resource-constrained embedded applications with hard real-time and low-power constraints. The segment-driven ADSW architecture improves the robustness of the matching process by utilizing adaptive support weight aggregation, offering an effective speed-accuracy tradeoff, however at the expense of high resource usage and processing of intermediate-resolution stereo images. Finally, the stereo vision architectures that integrate the GIF (in the cost aggregation and/or the disparity refinement steps) enable real-time and concurrently high-quality depth computation even at high resolution images. The thesis has demonstrated how the developed architectures can take advantage of the fine-grain parallelism made available by programmable hardware (FPGAs) to enable significant speedups and energy savings compared to traditional general-purpose computer based implementations.

However, the promise of FPGA technology extends beyond the implementation of the different architectures independently. The real potential of FPGAs lies in Adaptive Computing Systems (ACS) - systems that adapt and evolve in response to the changing environment while operational, without compromising the consistency and real-time properties of the system [**183**]. As such, a future directive of this thesis lies in the content of adaptive computing in order to create several different configurations of the developed stereo vision architectures, each tailored to the specific set of operational requirements mentioned above. This will enable the development of an ACS that can be reconfigured with a suitable configuration when the environment and thus the operational requirements change. In this way, the results obtained in the thesis can be utilized in the long-term to deliver real-time depth computation for a large number of applications targeting different operational goals, while maximizing component utilization and minimizing hardware redundancy.

#### 7.2.2.2 *Using Approximate Computing for Energy-Efficient Stereo Vision Designs*

Approximate Computing (AC) has emerged as a promising approach for the design of energy efficient dedicated hardware architectures in embedded environments. It refers to the ability of the architecture to tolerate some loss of quality in the computed results through approximate circuit design of arithmetic operations and algorithm-level techniques, with the aim to allow for substantially improved energy efficiency [**184**]. In Section 4, we discussed how a complex ADSW algorithm can be adopted for an efficient hardware design through a series of hardware-oriented arithmetic approximations and optimization techniques. The resulting system, however, was able to provide a specific set of operational requirements (accuracy, processing speed, energy efficiency, etc.) regardless of the application domain and working environment.

Consequently, a future improvement would be to utilize the approximate computing approach to endow the proposed stereo vision architectures with a capability similar to the human brain's ability to scale the degree of accuracy needed for a given task. The main idea is to explore the real-time reconfiguration aspect of FPGAs to use a specific set of hardware-oriented optimizations at any time depending on the actual state of the environment and the operational requirements. In this way, we can develop stereo vision architectures that do not compute the same level of accuracy all the time. Instead, the application context could be utilized to dictate different levels of effort, thus offering a scalable approach in terms of providing a desirable trade-off between efficiency (processing performance or energy) based on the content of the environment.

#### 7.2.2.3 *Optimization of the architectures to support next-generation image resolutions*

High resolution images offer a high pixel density, and therefore more details about the captured scene. Hence, the ability to process high resolution images is becoming important in computer vision applications for better accuracy in the analysis of images. Many applications require zooming of a specific area of interest in the image wherein high resolution becomes essential, e.g. surveillance, forensic and satellite imaging applications. Today, image sensor and display technology are evolving. Ultra High-Definition (UHD) and 4K (which deliver four times the picture resolution of 1080p Full HD, that's eight million pixels compared to two million pixels) are rewriting the rulebook when it comes to image quality and clarity, offering more fine detail, greater texture and an almost photographic emulsion of smoothness.

Depth estimation for such increased image resolutions is expected to play a significant role towards the development of future 3D TV applications. High resolution depth estimation would be of importance in pattern recognition, and also in medical imaging for diagnosis as well. This thesis has developed stereo vision architectures that can provide high-definition (including 720p and 1080p) depth computation (the edge-directed and GIF- based architectures). The hardware implementation complexity of the architectures and how they scale towards higher resolution images and image sizes were presented in the corresponding Sections (Sections 3 & 5). Considering the importance and potential of upcoming UHD and 4K images, a topic that deserves investigation in the near-future is the optimization of the proposed stereo vision architectures to support processing of next-generation image resolutions (UHD, 4K, or even 8K).

#### 7.2.2.4 *Fusion with other types of depth sensors*

Depth extraction using passive stereo technology is a well-studied problem. Advances in both algorithms and computing devices and platforms have enabled the development of real-time and accurate solutions for embedded environments. Especially, the stereo matching architecture presented in Chapter 5, which implements a local stereo algorithm, is capable of providing a matching accuracy of 93.6%, while the top performing algorithm in the Middlebury evaluation website achieves ~96%, based on global optimization [**26**]. The regions where local stereo algorithms still not work reliably include non-textured and featureless regions of a scene. In such cases, it is particularly challenging to establish correspondence as there is simply insufficient visual information across multiple cameras. Global stereo algorithms solve this problem by propagating information from textured to textureless regions, using disparity optimization (e.g. Belief propagation, graph cuts, etc.). However, they are known to be quite fragile in practice and slow [**71**].

A recent trend in depth estimation is the fusion of multiple sensors together, as a means to enable higher depth accuracy than can be achieved by each sensor individually. As discussed in Chapter 2.2, it is not trivial to choose a winner among the available depth sensing technologies, as the different depth sensors have their pros and cons, and can be more suitable than others in particular conditions and environments. On one hand, it is well understood that passive stereo sensors fail in non-textured, featureless portions of a scene, in regions with repetitive patterns and occluded areas, as these regions cause the generation of multiple local minima. On the other hand, active sensors are more accurate in these regions, but they currently have a low image resolution and tend to be noisy in highly textured regions for which stereo excels (due to the existence of unique local minima). Therefore, the main idea behind this trend is to take advantage of the complementary nature of more than one depth sensors in order to generate highly-accurate depth maps at all image regions.

Exploiting the complementary nature of different depth sensors is a recent research area that received attention during the last few years. Most of the existing approaches fuse stereo with Time-of-Flight (ToF) depth sensors. A comprehensive review can be found [**185**]. The work in [**186**] proposed fusing a laser range finder with a stereo vision camera for obstacle avoidance applications. Other approaches combine the Kinect with stereoscopic cameras [**187**]. The aforementioned implementations focus mostly on highly-accurate depth estimation, and most of them are not capable for real-time performance. In particular, among the implementations for which a processing performance is provided, the reported processing times are 8-18 fps for a resolution of 1024x768, while the work in [**187**] that targets ultra high definition images 94-12MP) requires 10-20 minutes to generate the combined depth maps.

As such, the work in this thesis which has focused on providing high-quality disparity estimation for resource constrained embedded systems can be continued by integrating another type of depth sensing technology, other than stereo vision, and use the provided depth maps, in order to improve the quality of the disparity maps produced by the architectures presented in this thesis. Some of the challenges involved in integrating another depth sensor into the existing architectures are listed below:

- Selecting a depth sensor that is affordable for embedded systems in terms of cost and energy consumption.
- Active depth sensors output much lower resolution than those from stereoscopic cameras. To address this problem, it will be necessary to develop real-time hardware designs for up-sampling the depth maps of the active sensors.
- Non-confident regions in the disparity maps generated by the proposed stereo matching architecture (e.g. textureless regions) will need to be indentified based on local image features.

#### 7.2.2.5 *Multi-view stereo*

Multi-view camera systems are becoming ubiquitous in today’s applications in stereoscopic 3D movie and broadcast production, video conferencing, surveillance, image-based rendering, etc. In such systems, several cameras are placed around the scene, which is captured from different points of view. Multi-view systems can provide a full 3D model of the scene, however, to do so, the images captured from the different cameras need to be correlated. This has to be done in real-time and under tight power envelops when targeting embedded applications.

In this thesis, we demonstrated how the use of edge information can significantly reduce the search space and enable high frame rates. The obtained frame rates might seem to high for most video processing applications, where 30 fps are enough. However, the real benefits of the edge directed stereo matching architecture can be illustrated in the case of multi-view stereo matching, where the same architecture can be shared over time (time-multiplexed) among multiple views keeping the overall frame rate at 30 fps, while performing correlation across multiple views without the need to replicate the architecture, thus also consuming less-energy.

#### 7.2.2.6 *A Machine Learning Approach to Depth Estimation: Integrating monocular depth cues with stereo vision*

Inspired by the way humans are able to perceive depth information from visual information (Chapter 2.3), relying also on monocular depth cues rather than only on binocular disparity (stereo vision), could definitely result in more robust computer vision based depth estimation systems. Humans have an amazing ability to judge depth even from a single image, by relying on monocular cues such as texture variations and gradients, occlusion, known object sizes, haze, defocus, etc. (see Chapter 2.3).

It is worth nothing that this approach is more efficient than fusing different depth sensors together, in that, monocular cues can be extracted from the individual images captured by the stereo camera. However, the real challenge here lies in determining what cues to extract and the difficulty of extracting depth cues from single images. For most depth cues, the global structure of the image needs to be account, as they rely on contextual information that represents global properties on an image and cannot be extracted from local patches. In addition, prior information about the scene is usually required. Local cues such as texture color and texture gradient may not be sufficient to accurately determine depth when using them alone, but they can potentially be of great value when combined with stereo disparity.

With the aforementioned considerations in mind, another future directive would be to capture monocular cues that can be extracted in real-time with an implementation based on reconfigurable hardware, and incorporate those cues into the proposed stereo vision systems so as to obtain better disparity estimates than the stereo matching architectures alone. Proper integration of monocular depth cues could result in more accurate depth map estimate (imitating human brain system). The most promising depth cues for a hardware implementation are motion parallax, texture variation, haze, perspective, vertical coordinate, and sharpness, which can be implemented based on local windows. These depth cues will result in a number of feature sets; the feature sets can be computed for different image-resolution levels to account also for different object sizes, (1, 1/2, and 1/4), to capture occlusion and to make these features global accountable features. In this way, depth estimation becomes a machine learning problem, where extracted features representing monocular depth cues are used to estimate depth. A recent software implementation that utilizes monocular depth cues to extract feature vectors that are used with a Support Vector Machine (SVC) is proposed in [**188**]. Investigating hardware design issues and considerations of this approach would certainly be of valuable importance in the embedded vision community.

## 7.3 Summary

Embedded vision has become an essential requirement for several emerging applications and markets in many industries including automotive, robotics, medical imaging, defense and many more. Empowering embedded vision systems with depth information extracted from stereoscopic vision can help overcome a variety of computer vision challenges, such as detecting obstacles in automotive and robotic applications or interpreting gestures for smart user interfaces. However, stereo vision algorithms typically require high compute performance, and, of course, their implementations for embedded vision systems usually need to fit into tight cost and power consumption envelopes. This thesis presented hardware architectures of stereo vision algorithms that have the potential to satisfy the requirements of constrained embedded vision applications. The developed architectures were evaluated in the applications of object detection and obstacle avoidance in robotics environments. The author expects that the results achieved in this thesis will be used long term by the embedded vision community, and will contribute significantly in the development of future 3D vision algorithms and applications.

# References

[1] Berkeley Design Technology, Inc. (2011) Implementing Vision Capabilities in Embedded Systems. [Online]. http://www.bdti.com/private/pubs/BDTI_ESC_Embedded_Vision.pdf

[2] ALTERA. (2012) Processing Options For Implementing Vision Capabilities in Embedded Systems. [Online]. http://www.altera.com/technology/system-design/articles/2012/vision-capabilities-in-embedded-systems.html

[3] T. Wilson and B. Dipert, "Embedded Vision on Mobile Devices," *Journal of Electronic Engineering*, July 2013.

[4] AVNET. (2013, June) EMBEDDED VISION: Creating a Next-Generation of Machines that "See". [Online]. http://www.em.avnet.com/en-us/design/publications/Documents/AXIOM_Embedded%20Vision.pdf

[5] Embedded Vision Alliance. (2014) Applications for Embedded Vision. [Online]. http://www.embedded-vision.com/applications/medical

[6] Jamie Hartford. (2013, April) The Embedded Vision Revolution. [Online]. http://www.mddionline.com/article/embedded-vision-revolution

[7] Argon Design. (2014, January) Embedded vision systems set to revolutionise electronics. [Online]. http://www.argondesign.com/news/2014/jan/22/embedded-vision-systems/

[8] A. Nieto, D. L. Vilarino, and V. B. Sánchez, "Towards the Optimal Hardware Architecture for Computer Vision," in *Machine Vision - Applications and System*.: INTECH, 2012, p. 27 pages.

[9] B. Cyganek and J. P. Siebert, *Introduction to 3D Computer Vision Techniques and Algorithms*.: Wiley, John & Sons, 2009.

[10] O. Faugeras, *Three Dimensional Computer Vision*.: MIT Press, 1993.

[11] M. Domínguez-Morales, A. Jiménez-Fernández, R. Paz-Vicente, A. Linares-Barranco, and G. Jiménez-Moreno, "Stereo Matching: From the Basis to Neuromorphic Engineering," in *Current Advancements in Stereo Vision*.: InTech, 2012.

[12] F. Cheng and X. Chen, "Integration of 3D stereo vision measurements in industrial robot applications," in *IAJC-IJME 2008 Conference*, Tennessee, 2008.

[13] (2013) RobotCar UK - Robotics Science For Smarter Cars. [Online]. http://mrg.robots.ox.ac.uk/robotcar/

[14] (2014) Mercedes-Benz S-class 2014. [Online]. http://www.mbusa.com/vcm/MB/DigitalAssets/pdfmb/brochures/2014-S-Class.pdf

[15] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed.: Addison-Wesley Pub (Sd), 2007.

[16] (2008) Bumblebee2 stereo vision camera. [Online]. www.ptgrey.com/products/Point_Grey_stereo_catalog.pdf

[17] I., Heikkinen, T., Myllyla, R. & Kilpela, A. Moring, "Acquisition of three-dimensional image data by a scanning laser range finder," *Optical Engineering*, vol. 28, no. 8, pp. 897-902, 1989.

[18] S.B. Gokturk, H. Yalcin, and C. Bamji, "A Time-Of-Flight Depth Sensor - System Description, Issues and Solutions," in *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW '04)*, vol. 4, 2004, pp. 35-43.

[19] B. Freedman, A. Shpunt, M. Machline, and Y. Arieli, "Depth mapping using projected patterns," Application US20080240502 A1, October 2, 2008.

[20] R. Szeliski, *Computer Vision: Algorithms and Applications*.: Spinger, 2010.

[21] P. R.a Sanz, B. R. Mezcua, and J. M. S. Pena, "Depth Estimation - An Introduction," in *Current Advancements in Stereo Vision*, Asim Bhatti, Ed.: INTECH, 2012, ch. 5.

[22] Reynold Bailey; Cindy Grimm; Christopher Davoli, "The effect of warm and cool object colors on depth ordering," in *APGV '06 Proceedings of the 3rd symposium on Applied perception in graphics and visualization*, New York, NY, USA, 2006, p. 161.

[23] A. Saxena, S. H. Chung, and A. Y. Ng, "3-D Depth Reconstruction from a Single Still Image," *International Journal of Computer Vision*, vol. 76, pp. 53-69, 2008.

[24] M. Bleyer and M. Gelautz, "A layered stereo matching algorithm using image segmentation and global visibility constraints," *Journal of Photogrammetry & Remote Sensing*, vol. 59, pp. 128-150, 2005.

[25] D. Szeliski and R. Scharstein, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Inter. J. of Comput. Vision*, vol. 47, pp. 7-42, 2002.

[26] (2010-2013) Stereo - The Middlebury Computer Vision Pages. [Online]. http://vision.middlebury.edu/stereo/

[27] enpeda. Image Sequence Analysis Test Site. [Online]. http://www.mi.auckland.ac.nz/EISATS/

[28] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *IEEE Conference on Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3354-3361.

[29] T. Vaudrey, C. Rabe, R. Klette, and J. Milburn, "Differences between stereo and motion behaviour on synthetic and real-world stereo sequences," in *23rd Int. Conf. on Image and Vision Computing (IVCNZ 2008)*, New Zealand, 26-28 Nov. 2008, pp. 1-6.

[30] D. G. Cubber, L. Nalpantidis, G. Sirakoulis, and A. Gasteratos, "Intelligent Robots need Intelligent Vision: Visual 3D Perception," in *IARP/EURON Workshop on Robotics for Risky Interventions and Environmental Surveillance*, 2008.

[31] R.A El-laithy, H. Jidong, and M. Yeh, "Study on the use of Microsoft Kinect for robotics applications," in *2012 IEEE/ION Position Location and Navigation Symposium (PLANS)*, 23-26 April 2012, pp. 1280-1288.

[32] M. Hansard, S. Lee, O. Choi, and R. Horaud, *Time-of-Flight Cameras: Principles, Methods and Applications*.: Series: Springer Briefs in Computer Science, 2013.

[33] S. Foix, G. Alenya, and C. Torras, "Lock-in Time-of-Flight (ToF) Cameras: A Survey," *IEEE Sensors Journal*, vol. 11, no. 9, pp. 1917-1926, Sept. 2011.

[34] Rama Rao Nidamanuri, P Bhanu Prasad Dhanya S Pankaj, "3-D Imaging Techniques and Review of Products," in *International Conference on "Innovations in Computer Science and Engineering (ICICSE 2013)*, At Hyderabad, India, 2013.

[35] PDMTechnologies. (2014) pmd[vision]® CamCube 3.0. [Online]. http://www.pmdtec.com/

[36] (2010) Microsoft Kinect. [Online]. http://www.xbox.com/en-us/kinect/

[37] ASUSTeK Computer Inc. (2014, March) Xtion PRO. [Online]. http://www.asus.com/Multimedia/Xtion_PRO/

[38] M. Antunes, J.P. Barreto, C. Premebida, and U. Nunes, "Can stereo vision replace a Laser

Rangefinder?," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 5183-5190.

[39] K. Konolige et al., "Outdoor mapping and navigation using stereo vision," in *Proc. of the Intl. Symp. on Experimental Robotics (ISER)*, 2006.

[40] K. Khoshelham, "Accuracy analysis of kinect depth data," in *ISPRS workshop laser scanning*, vol. 38, Calgary Canada, 2011, pp. 133-138.

[41] R. Nair et al., "High Accuracy TOF and Stereo Sensor Fusion at Interactive Rates," in *Computer Vision – ECCV 2012. Workshops and Demonstrations, Lecture Notes in Computer Science*.: Springer Berlin Heidelberg, 2012, pp. 1-11.

[42] Stephan Reichelt, RalfHäussler,Gerald Fütterer, and Norbert Leister, "Depth cues in human visual perception and their realization in 3D displays," in *Three-Dimensional Imaging, Visualization, and Display 2010 and Display Technologies and Applications for Defense, Security, and Avionics*, 2010, p. 12.

[43] Wikipedia. (2014, March) Pinhole camera --- Wikipedia

[44] E. Staudinger, M. Humenberger, and W. Kubinger, "FPGA-based Rectification and Lens Undistortion for a Real-Time Embedded Stereo Vision Sensor," in *Proc. FH Science Day 2008*, Linz, Austria, 2008, pp. 18-25.

[45] E. Trucco and A. Verri, *Introductory Techniques For 3-D Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[46] Y.M Wang, Y. L, and J. B. Zheng, "A camera calibration technique based on OpenCV," in *2010 3rd International Conference on Information Sciences and Interaction Sciences (ICIS)*, 23-25 June 2010, pp. 403-406.

[47] J.-Y. Bouguet. (2008, January) Camera Calibration Toolbox for Matlab. [Online]. http://www.vision.caltech.edu/bouguetj/calib_doc/index.html

[48] M. Warren, D. McKinnon, and B. Upcroft, "Online Calibration of Stereo Rigs for Long-Term Autonomy," in *International Conference on Robotics and Automation*, Karlsruhe, Germany , 2013.

[49] T. Dang, C. Hoffmann, and C. Stiller, "Continuous Stereo Self-Calibration by Camera Parameter Tracking," *IEEE Transactions on Image Processing*, vol. 18, no. 7, pp. 1536-1550, July 2009.

[50] D. V. Papadimitriou and T. J. Dennis, "Epipolar line estimation and rectification for stereo image pairs," *IEEE Trans. on Image Processing*, vol. 2, pp. 672-676, August 2005.

[51] C. Loop and Z. Zhengyou, "Computing rectifying homographies for stereo vision," in *1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1999, p. 131.

[52] B. Maldeniya, D. Nawarathna, K. Wijayasekara, T. Wijegoonasekara, and R. Rodrigo, "Computationally efficient implementation of video rectification in an FPGA for stereo vision applications," in *2010 5th International Conference on Information and Automation for Sustainability (ICIAFs)*, 17-19 Dec. 2010, pp. 219-224.

[53] C. Vancea and S. Nedevschi, "LUT-based Image Rectification Module Implemented in FPGA," in *2007 IEEE International Conference on Intelligent Computer Communication and Processing*, 6-8 Sept. 2007, pp. 147-154.

[54] K. Jawed, J. Morris, T. Khan, and G. Gimel'farb, "Real Time Rectification for Stereo Correspondence," in *2009 International Conference on Computational Science and Engineering*, 2009, pp. 277-284.

[55] D. H. Park, H. S. Ko, J. G. Kim, and J. D Cho, "Real time rectification using differentially encoded lookup table," in *Proceedings of the ACM 5th international Conference on Ubiquitous information Management and Communication (ICUIMC '11)*, Seoul, Korea, February 21 - 23, 2011, 2011, pp. 1-4.

[56] T. Kanade, H. Kano, S. Kimura, A. Yoshida, and K. Oda, "Development of a video-rate stereo machine," in *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots*, 9 Aug 1995, pp. 95-100.

[57] O. Faugeras et al., "Real time correlation-based stereo: algorithm, implementations and applications," Technical Report 2013, INRIA Sophia Antipolis, 1993.

[58] A. Ansar, A. Castano, and L. Matthies, "Enhanced real-time stereo using bilateral filtering," in *2nd International Symposium on3D Data Processing, Visualization and Transmission, 2004 (3DPVT 2004)*, 6-9 Sept. 2004, pp. 455-462.

[59] H. Hirschmuller and D. Scharstein, "Evaluation of Stereo Matching Costs on Images with Radiometric Differences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 9, pp. 1582-1599, Sept. 2009.

[60] M. Z. Brown, D. Burschka, G. D. Hager, and S. Member, "Advances in computational stereo," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 993-1008, 2003.

[61] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Classification and evaluation of cost aggregation methods for stereo correspondence," in *IEEE Int. Conf. Comput. Vision Pattern Recognit.*, Jun. 2008, pp. 1-8.

[62] K.-J. Yoon and I.-S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650-656, Apr. 2006.

[63] F. Tombari, S. Mattoccia, and L. Di Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," *Lecture Notes in Computer Science*, vol. 4872, pp. 427–438., Dec. 2007.

[64] K., Maier, D., Hesser, J., Manner, R. Muhlmann, "Calculating dense disparity maps from color stereo images, an efficient implementation," in *Proceedings of the IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, 2001, pp. 30-36.

[65] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, "A fast stereo matching algorithm suitable for embedded real-time systems," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1180-1202, November 2010.

[66] K. Zhang, J. Lu, and G. Lafruit, "Cross-Based Local Stereo Matching Using Orthogonal Integral Images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 7, pp. 1073-1079, July 2009.

[67] C. Richardt, D. A. H. Orr, I. P. Davies, A. Criminisi, and N. A. Dodgson, "Real-time Spatiotemporal Stereo Matching Using the Dual-Cross-Bilateral Grid," in *European Conference on Computer Vision (ECCV)*, Heraklion, Crete, Greece, 5-11 Sept. 2010, pp. 510-523.

[68] S. Perri, P. Corsonello, and G. Cocorullo, "Adaptive Census Transform: A novel hardware-oriented stereovision algorithm," *Computer Vision and Image Understanding*, vol. 117, no. 1, pp. 29-41, January 2013.

[69] H. Hirschmüller, "Stereo Processing by Semiglobal Matching and Mutual Information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328-341, 2008.

[70] P. Steingrube, S. K. Gehrig, and U. Franke, "Performance Evaluation of Stereo Algorithms for Automotive Applications," in *Lecture Notes in Computer Science*, vol. 5815, 2009, pp. 285-294.

[71] B. Tippetts, D. J. Lee, K. Lillywhite, and J. Archibald, "Review of stereo vision algorithms and their suitability for resource-limited systems," *Journal of Real-Time Image Processing*, 2013.

[72] Z. Yuhang, R. Hartley, J. Mashford, and S. Burn, "Superpixels, Occlusion and Stereo," in *2011 International Conference on Digital Image Computing Techniques and Applications (DICTA)*, 6-8 Dec. 2011, pp. 84-91.

[73] Q. Yang, C. Engels, and A. Akbarzadeh, "Near Real-time Stereo for Weakly-Textured Scenes," in *British Machine Conference*, September 2008, pp. 72.1-72.10.

[74] C. Wöhler and P. d'Angelo, "Stereo Image Analysis of Non-Lambertian Surfaces," *International Journal of Computer Vision*, vol. 81, no. 2, pp. 172-190, 2009.

[75] W. Zhao and N. Nandhakumar, "Effects of camera alignment errors on stereoscopic depth estimates," *Pattern Recognition, Elsevier*, vol. 29, no. 12, pp. 2115-2126, Dec. 1996.

[76] Point Grey Research Inc. (2014, April) Point Grey - Innovation in Imaging. [Online]. http://ww2.ptgrey.com/

[77] Point Grey Research Inc., "Stereo Vision Introduction and Applications ," Point Grey Research Inc., Canada, Technical Application Note 2010.

[78] DARPA. (2014, March) DARPA URBAN CHALLENGE. [Online]. http://archive.darpa.mil/grandchallenge/

[79] Aldebaran Robotics. (2014, March) Aldebaran Robotics. [Online]. http://www.aldebaran.com/en

[80] Aldebaran Robotics. (2014, March) The Companion Robot NAO. [Online]. https://team.inria.fr/perception/nao/

[81] NASA. (2014, April) Mars Science Laboratory. [Online]. http://www.nasa.gov/mission_pages/msl/#.U1Z-DPl_tGQ

[82] Ben Coxworth. (2011, May) Continental forward braking system to get stereo vision. [Online]. http://www.gizmag.com/continental-forward-braking-system-stereo-vision/18594/

[83] R. Kalarot and J. Morris, "Comparison of FPGA and GPU implementations of real-time stereo vision," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 13-18 June 2010, pp. 9-15.

[84] W. James MacLean, "An Evaluation of the Suitability of FPGAs for Embedded Vision Systems," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer vision and pattern recognition*, San Diego, CA, 2005, p. 131.

[85] L. Nalpantidis, G. C. Sirakoulis, and A. Gasteratos, "Review of stereo vision algorithms: from software to hardware," *Inter. J. of Optomechatronics*, vol. 2, no. 4, pp. 435-462, 2008.

[86] C. Banz, H. Blume, and P. Pirsch, "Architectures for Stereo Vision," in *Handbook of Signal Processing Systems*.: Springer New York, 2013, pp. 483-515.

[87] H. Hirschmuller, P.R. Innocent, and J.Garibaldi, "Real-time correlation-based stereo vision with reduced border errors," *Int. J. of Computer Vision*, vol. 47, no. 1-3, pp. 229-246, April-June 2002.

[88] S. Forstmann, Y. Kanou, Jun Ohya, S. Thuering, and A. Schmitt, "Real-Time Stereo by using Dynamic Programming," in *Proc. of Computer Vision and Pattern Recognition Workshop (CVPRW '04)*, June 2004, pp. 29-29.

[89] C., Humenberger, M. and Ambrosch, K. Zinner, "An Optimized Software-Based Implementation of a Census-Based Stereo Matching Algorithm," in *Advances in Visual Computing, Lecture*

*Notes in Computer Science*.: Springer Berlin Heidelberg, 2008, vol. 5358, pp. 216-227.

[90] S.K. Gehrig and C. Rabe, "Real-time Semi-Global Matching on the CPU," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2010, pp. 85-92.

[91] R. Yang and M. Pollefeys, "A versatile stereo implementation on commodity graphics hardware," *Real-Time Imaging*, vol. 11, no. 1, pp. 7-18, Feb. 2005.

[92] Q. Yang et al., "Real-time global stereo matching using hierarchical belief propagation," in *The British Machine Vision Conf.*, 2006.

[93] Liang Wang, Miao Liao, Minglun Gong, Ruigang Yang, and D. Nister, "High-Quality Real-Time Stereo Using Adaptive Cost Aggregation and Dynamic Programming," in *Third International Symposium on 3D Data Processing, Visualization, and Transmission*, 2006, pp. 798-805.

[94] Xueqin Xiang, Mingmin Zhang, Guangxia Li, Yuyong He, and Zhigeng Pan, "Real-time stereo matching based on fast belief propagation," *Machine Vision and Applications*, vol. 23, no. 6, pp. 1219-1227, November 2012.

[95] I. Ernst and H. Hirschmüller, "Mutual Information Based Semi-Global Stereo Matching on the GPU," in *Advances in Visual Computing, Lecture Notes in Computer Science*., 2008, vol. 5358, pp. 228-239.

[96] Alan Gray. (2012) http://www2.epcc.ed.ac.uk/~alang/GPU_training_aug12/GPU_Architecture.pdf.

[97] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A Performance Energy Comparison of FPGAs, GPUs, and Multicores for Sliding-Window Applications," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA'12)*, New York, NY, USA, 2012, pp. 47-56.

[98] K. Konolige, "Small vision system - hardware and implementation," in *Proc. of the International Symposium on Robotics Research*, Hayama, Japan, 1997, pp. 111-116.

[99] N. Chang, Ting-Min Lin, Tsung-Hsien Tsai, Yu-Cheng Tseng, and Tian-Sheuan Chang, "Real-Time DSP Implementation on Local Stereo Matching," in *IEEE International Conference on Multimedia and Expo*, 2-5 July 2007, pp. 2090-2093.

[100] R.-P M. Berretty, A. K. Riemens, and P. F. Machado, "Real-time embedded system for stereo video processing for multiview displays," in *Stereoscopic Displays and Virtual Reality Systems XIV*, San Jose, USA, Jan. 2007.

[101] B. Khaleghi, S. Ahuja, and Q. Wu, "An improved real-time miniaturized embedded stereo vision system (MESVS-II)," in *IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Workshops 2008 (CVPRW '08)*, 23-28 June, 2008, pp. 1-8.

[102] S. Cavanag and M. Manzke, "Real time disparity map estimation on the cell processor," in *Proceedings of the 2009 Eurographics Ireland Workshop*, Trinity College Dublin, December, 2009, pp. 67-74.

[103] J. Liu et al., "Disparity map computation on a cell processor," in *Proceedings of the IASTED International Conference on Modelling, Simulation, and Identification (MSI '09)*, Beijing, China, 12-14 October, 2009.

[104] Kun-Yuan Hsieh, Chi-Hua La, Shang-Hong Lai , and Jenq Kuen Lee, "Parallelization of Belief Propagation on Cell Processors for Stereo Vision," *ACM Trans. Embed. Comput. Syst.*, vol. 11S, no. Article 13, p. 15 pages, June 2012.

[105] C. Georgoulas and I. Andreadis, "A Real-Time Occlusion Aware Hardware Structure for Disparity Map Computation," *Image Analysis and Process. – ICIAP 2009*, vol. 5716, pp. 721-730, 2009.

[106] K. Ambrosch, M. Humenberger, W. Kubinger, and A. Steininger, "A SAD-based Stereo Matching Using FPGAs," in *Embedded Computer Vision part II*. London: Spinger, 2009, pp. 121-138.

[107] A. Darabiha, J. MacLean, and J. Rose, "Reconfigurable hardware implementation of a phase-correlation stereo algorithm," *Machine Vision Appl.*, vol. 17, no. 2, pp. 116-132, 2006.

[108] S. Jin et al., "FPGA design and implementation of a real-time stereo vision system," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15-26, 2010.

[109] K. Ambrosch and W. Kubinger, "Accurate hardware-based stereo vision," *Comput. Vis. Image Und.*, vol. 114, no. 11, pp. 1303-1316, Nov. 2010.

[110] S. K. Gehrig, F. Eberli, and T. Meyer, "A Real-Time Low-Power Stereo Vision Engine Using Semi-Global Matching," in *Computer Vision Systems, Lecture Notes in Computer Science*.: Springer Berlin Heidelberg, 2009, vol. 5815, pp. 134-143.

[111] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation," in *2010 International Conference on Embedded Computer Systems (SAMOS)*, 2010, pp. 93-101.

[112] C. Banz, S. Hesselbarth, H. F. H. Blume, and P. Pirsch, "Real-Time Stereo Vision System using Semi-Global Matching Disparity Estimation: Architecture and FPGA-Implementation," *Transactions on HiPEAC*, vol. 5, no. 4, 2011.

[113] H. Hirschmüller, M. Buder, and I. Ernst, "MEMORY EFFICIENT SEMI-GLOBAL MATCHING," in *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci*, vol. I-3, 2012, pp. 371-376.

[114] N. Chang, T. Tsai, B. Hsu, Y. Chen, and T. Chang, "Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 20, no. 6, pp. 792-805, 2010.

[115] J. Ding et al., "Real-time stereo vision system using adaptive weight cost aggregation approach," *EURASIP JIVP*, vol. 2011, no. 1, pp. 1-19, December 2011.

[116] H. Kim and K. Sohn, "3D Reconstruction of Stereo Images for Interaction between Real and Virtual Worlds," in *2nd IEEE/ACM Int. Symposium on Mixed and Augmented Reality*, October 2003, pp. 169-176.

[117] J. Ralli, F. Pelayo, and J. Diaz, "Increasing Efficiency in Disparity Calculation," in *2nd Int. Conf. on Advances in brain, vision, and artificial intelligence*, vol. 4729, 2007, pp. 298-307.

[118] M. Raman and H. Aggarwal, "Study and Comparison of Various Image Edge Detection Techniques," *Int. J. of Image Process.*, vol. 3, no. 1, pp. 1-12, 2009.

[119] Z. Vasicek and L. Sekanina, "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *Int. J. of Innovative Computing and Applications*, vol. 1, no. 1, pp. 63-73, 2007.

[120] M. Hariyama, Y. Kobayashi, H. Sasaki, and M. Kameyama, "FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture," in *Proc. of the 48th Midwest Symposium on Circuits and Systems*, vol. 2, Covington, KY, 2005, pp. 1219 – 1222.

[121] Xilinx Inc. (2010, May) Virtex-5 LXT FPGA ML505 Evaluation Platform. [Online].

http://www.xilinx.com/products/devkits/HW-V5-ML505-UNI-G.htm

[122] Xilinx Inc. (2010, May) MicroBlaze Soft Processor Core. [Online]. http://www.xilinx.com/tools/microblaze.htm

[123] H. Hile and C. Zheng, "Stereo Video Processing for Depth Map," University of Washington, 2004.

[124] Y. Miyajima and T. Maruyama, "A Real-Time Stereo Vision System with FPGA," in *Proc. of the 13th Int. Conf. on Field-Programmable Logic and Applications (FPL 2003)*, Lisbon, Portugal, 2003, pp. 448-457.

[125] M. Arias-Estrada and J. M. Xicotencatl, "Multiple Stereo Matching Using an Extended Architecture," in *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, Ed., 2001, vol. 2147, pp. 203-212.

[126] J. Diaz, E. Ros, R. Carrillo, and A. Prieto, "Real-time system for high-image resolution disparity estimation," *IEEE Trans. on Image Process.*, vol. 16, no. 1, pp. 280–285, Jan. 2007.

[127] C. Kyrkou, C. Ttofis, and T. Theocharides, "Depth-Directed Hardware Object Detection," in *Proc. of the DATE'11*, Grenoble, France, 2011.

[128] S. H. Lee, J. Yi, and J. Kim, "Real-Time Stereo Vision on a Reconfigurable System," in *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, Ed., 2005, vol. 3553, pp. 299–307.

[129] Asmaa Hosni, Michael Bleyer, and Margrit Gelautz, "Secrets of adaptive support weight techniques for local stereo matching," *Computer Vision and Image Understanding*, vol. 117, no. 6, pp. 620-632, June 2013.

[130] J. Fang et al., "Accelerating Cost Aggregation for Real-Time Stereo Matching," in *IEEE 18th International Conference on Parallel and Distributed Systems*, Singapore, 17-19 December, 2012, pp. 472-481.

[131] S.A. Fahmy, P.Y.K. Cheung, and W. Luk, "Novel FPGA-based implementation of median and weighted median filters for image processing," in *International Conference on Field Programmable Logic and Applications*, 2005, pp. 142-147.

[132] H. Sunyoto, W. van der Mark, and D.M. Gavrila, "A comparative study of fast dense stereo vision algorithms," in *Intelligent Vehicles Symposium, 2004 IEEE*, 14-17 June 2004, pp. 319-324.

[133] TOKYO ELECTRON DEVICE LIMITED. (2013) Kintex-7 FPGA Display Kit. [Online]. http://solutions.inrevium.com/products/kits/consumer/tb-7k-acdc.html

[134] Xilinx Inc. (2013) Kintex-7 FPGA Family. [Online]. http://www.xilinx.com/products/silicon-devices/fpga/kintex-7/

[135] TOKYO ELECTRON DEVICE LIMITED. (2010) FMC Cards: Image Processing. [Online]. http://solutions.inrevium.com/products/fmc/image_processing/index.html

[136] Nadia Baha and Slimane Larabi, "ACCURATE REAL-TIME DISPARITY MAP COMPUTATION BASED ON VARIABLE SUPPORT WINDOW," *International Journal of Artificial Intelligence & Applications*, vol. 2, no. 3, July 2011.

[137] N. Baha and S. Larabi, "Accurate real-time neural disparity MAP estimation with FPGA," *Pattern Recognition*, vol. 45, no. 3, pp. 1195-1204, March 2012.

[138] S. K. Gehrig, F. Eberli, and T. Meyer, "A Real-Time Low-Power Stereo Vision Engine Using Semi-Global Matching," *Computer Vision Systems, Lecture Notes in Computer Science*, vol.

5815, pp. 134-143, 2009.

[139] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga-implementation," in *ICSAMOS*, 2010, pp. 93-101.

[140] L. Zhang et al., "Real-Time High-Definition Stereo Matching on FPGA," in *FPGA'11*, Monterey, CA, USA, 2011, pp. 55-64.

[141] Kaiming He, Jian Sun, and Xiaoou Tang, "Guided Image Filtering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 6, pp. 1397-1409, 2013.

[142] A. Hosni, M. Bleyer, C. Rhemann, M. Gelautz, and C. Rother, "Real-time local stereo matching using guided image filtering," in *IEEE Int. Conf. on Multimedia and Expo (ICME)*, 2011, pp. 1-6.

[143] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu, "Constant time Weighted Median Filtering for Stereo Matching and Beyond," in *International Conference on Computer Vision*, 2013.

[144] S. Perreault and P. Hebert, "Median Filtering in Constant Time," *IEEE Trans. on Image Process.*, vol. 16, no. 9, pp. 2389-2394, 2007.

[145] H.-S. Son, K.-r. Bae, S.-H. Ok, Y.-H. Lee, and B. Moon, "A Rectification Hardware Architecture for an Adaptive Multiple-Baseline Stereo Vision System," in *Communication and Networking*.: Springer Berlin Heidelberg, 2012, pp. 147-155.

[146] M. S. Sadri et al, "An FPGA Based Fast Face Detector," in *Global Signal Processing Expo and Conf.*, 2004.

[147] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using haar classifiers," in *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, New York, NY, USA, 2009, pp. 103-112.

[148] R. McCready, "Real-Time Face Detection on a Configurable Hardware System," in *10th International Workshop on Field-Programmable Logic and Applications*, 2000, pp. 157-162.

[149] M. Hiromoto, H. Sugano, and R. Miyamoto, "Partially Parallel Architecture for Adaboost-Based Detection with Haar-Like Features," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 19, no. 1, pp. 41-52, Jan. 2009.

[150] T. Darrell, G. Gordon, M. Harville, and J. Woodfill, "Integrated Person Tracking Using Stereo, Color, and Pattern Detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1998, p. 601.

[151] Shireen, Khaled, and Sumaya, "Moving object detection in spatial domain using background removal techniques - state-of-art," *Recent Patents on Computer Science*, vol. 1, pp. 32-34, 2008.

[152] H. Wu, K. Suzuki, T. Wada, and Q. Chen, "Accelerating Face Detection by Using Depth Information," in *3rd Pacific Rim Symposium on Advances in Image and Video Technology*, Japan, Jan. 2009, pp. 13-16.

[153] S. Kosov, K. Scherbaum, K. Faber, T. Thormahlen, and H-P Seidel, "Rapid stereo-vision enhanced face detection," in *16th IEEE International Conference on Image Processing*, 2009, pp. 1221-1224.

[154] Y.G. Wang, E.T. Lim, and R. Venkateswarlu, "Stereo head/face detection and tracking," in *International Conference on Image Processing*, vol. 1, 2004, pp. 605-608.

[155] C. Kyrkou and T. Theocharides, "SCoPE: Towards a Systolic Array for SVM Object Detection,"

*IEEE Embedded System Letters*, vol. 1, no. 2, pp. 46-49, August 2009.

[156] J. Kovac, P. Peer, and F. Solina, "Human Skin Color Clustering for Face Detection," in *EUROCON International Conference on Computer as Tool*, vol. 2, Sept. 2013, pp. 144-148.

[157] Lin Hwei-Jen, Yen Shwu-Huey, Yeh Jih-Pin, and Lin Meng-Ju, "Face Detection Based on Skin Color Segmentation and SVM Classification," in *Second International Conference on Secure System Integration and Reliability Improvement (SSIRI'08)*, July 2008, pp. 230-231.

[158] He Chun, A. Papakonstantinou, and Chen Deming, "A novel SoC architecture on FPGA for ultra fast face detection," in *IEEE International Conference on Computer Design, 2009 (ICCD 2009)*, Oct. 2009, pp. 412-418.

[159] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121-167, 1998.

[160] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: an application to face detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 1997, pp. 130-136.

[161] B. Heisele, T. Poggio, and M. Pontil, "Face Detection in Still GrayImages," in *unpublished*.

[162] MIT Center for Biological and Computation Learning. (2010, Jan.) CBCL Face Database #1. [Online]. Available: http://cbcl.mit.edu/software-datasets/FaceData2.html

[163] L. Nalpantidis and A. Gasteratos, "Stereo Vision Depth Estimation Methods for Robotic Applications," in *Depth Map and 3D Imaging Applications: Algorithms and Technologies*.: IGI Global, 2012, pp. 397-417.

[164] R. P. Sadolikar and P. C. Bhaskar, "Obstacle Detection and Avoidance using Stereo Vision System with Region of Interest (ROI) on FPGA," *International Journal of Engineering Research and Technology*, vol. 3, no. 3, March 2014.

[165] S. Mattoccia, "Stereo Vision Algorithms for FPGAs," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2013, pp. 636-641.

[166] J. Borenstein and Y. Koren, "Obstacle avoidance with ultrasonic sensors," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 213–218, 1988.

[167] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots in cluttered environments," in *IEEE International Conference on Robotics and Automation*, vol. 1, 1990, pp. 572–577.

[168] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[169] D. Sales, D. Correa, F. S. Osorio, and D. F. Wolf, "3D Vision-Based Autonomous Navigation System Using ANN and Kinect Sensor," in *13th International Conference, EANN 2012*, London, UK, Sept. 2012, pp. 305-314.

[170] L. Nalpantidis and A. Gasteratos, "Stereovision-based fuzzy obstacle avoidance method," *International Journal of Humanoid Robotics*, vol. 8, no. 1, pp. 169-183, 2011.

[171] T. Pire, P. De Cristoforis, M. Nitsche, and J. J. Berlles, "Stereo vision obstacle avoidance using depth and elevation maps," in *IEEE VI RAS Summer School on "Robot Vision and Applications"*, Santiago, Chile, Dec. 2012.

[172] J. L. Crowley, "World modeling and position estimation for a mobile robot using ultrasonic ranging," in *IEEE Int. Conf. on Robotics and Automation*, vol. 2, 1989, pp. 674-680.

[173] O. Khatib, "Motion coordination and reactive control of autonomous multi-manipulator system," *Journal of Robotic Systems*, vol. 15, no. 4, pp. 300–319, 1996.

[174] O. Khatib, "Robot in human environments: Basic autonomous capabilities," *International Journal of Robotics Research*, vol. 18, no. 7, pp. 684–696, 1999.

[175] LeapMotion. [Online]. https://www.leapmotion.com/

[176] D.S.O. Correa et al., "Mobile Robots Navigation in Indoor Environments Using Kinect Sensor," in *2012 Second Brazilian Conference Critical Embedded Systems (CBSEC)*, May 2012, pp. 36-41.

[177] A. Ohya, A. Kosaka, and A. Kak, "Vision-based navigation of mobile robot with obstacle avoidance by single camera vision and ultrasonic sensing," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 969–978, 1998.

[178] L. Nalpantidis, I. Kostavelis, and A. Gasteratos, "Stereovision-based algorithm for obstacle avoidance," in *International Conference on Intelligent Robotics and Applications: Lecture Notes in Computer Science, Springer Berlin Heidelberg*, 2009, pp. 195-204.

[179] Digilent Atlys. [Online]. www.digilentinc.com/atlys/

[180] Raspberry Pi. [Online]. http://www.raspberrypi.org/

[181] Arduino. [Online]. http://www.arduino.com/

[182] FPGALink Library. [Online]. http://www.makestuff.eu/wordpress/software/fpgalink/

[183] Ming Liu, "Adaptive Computing based on FPGA Run-time Reconfigurability," Royal Institute of Technology, Stockholm, PhD Thesis 978-91-7415-985-1 / 1653-6363, 2011.

[184] H. Jie and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)* , 27-30 May 2013, pp. 1-6.

[185] R Nair et al., "A Survey on Time-of-Flight Stereo Fusion," in *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications: Lecture Notes in Computer Science*., 2013, vol. 8200, pp. 105-127.

[186] S. Kumar, D. Gupta, and S. Yadav, "Sensor fusion of laser & stereo vision camera for depth estimation and obstacle avoidance," *International Journal of Computer Applications*, vol. 1, no. 25, pp. 20-25, Feb. 2010.

[187] G. Somanath, S. Cohen, B. Price, and C. Kambhamettu, "Stereo+Kinect for High Resolution Stereo Correspondences," in *2013 International Conference on 3D Vision (3DV'13)*, Washington, DC, USA, 2013, pp. 9-16.

[188] Zhipeng Fan, Mingjun Li, and Ying Lu, "An Efficient Image Depth Extraction Method Based on SVM," *International Journal of Multimedia and Ubiquitous Engineering* , vol. 8, no. 3, May 2013.

# APPENDIX A

# Case Study: Edge-Accelerated Adaptive Support Weight Stereo Matching Design

*This section presents the design of a stereo matching design that uses image segmentation and adaptive support weights. Although it has a similar structure with the architecture presented in Chapter 4, the architecture presented here also utilizes information extracted by an edge detection mechanism, in an attempt to reduce the search space over which the matching process is executed. By combining features from both Chapters 3 & 4, this section aims to illustrate how edge information can been utilized in ADSW-based hardware designs to enable the realization of complex stereo matching algorithms on reconfigurable hardware. The resulting stereo matching design was implemented on a Virtex-5 FPGA platform, yielding real-time performance (30 frames per second – fps) for a stereo image pair of 640x480. Moreover, the architecture is able to improve the quality of the disparity maps compared to other hardware implementations that feature fixed support local stereo correspondence algorithms.*

## 9.1 Algorithm Overview

The algorithm's steps towards computing a disparity value for each pixel $p$ in $I_r$ are given in Fig. 1. The algorithm starts by determining whether a pixel $p$ corresponds to an edge or not. For each pixel $p$ corresponding to an edge, the algorithm extracts an *mxm* support window $W_r$ centered on $p$ in $I_r$, and an *mxm* support window $W_t$ centered on $q$ in $I_t$ (the coordinate of $q$ is ($x$+$d$, $y$), where $d$ lies in the range $d_m$ to $d_M$). The algorithm then applies a segmentation process over $W_r$ and $W_t$, and uses the information obtained by the segmentation in the weight generation step, which generates the weight coefficients $w'_r$ and $w'_t$ (for each pixel falling in $W_r$ and $W_t$) as in (1), where $S_c$ is the segment where the central point $p_c$ (or $q_c$) of the support window $W_r$ (or $W_t$) lies, $d_c$ is the Euclidean distance between two triplets in the CIELAB color space, and $\gamma_c$ is a parameter of the algorithm. The next step is to compute a pointwise score for any pixel $p_i \in W_r$ corresponding to $q_i \in W_t$. The pointwise scores, which are selected as the Absolute Difference (AD) of $p_i$ and $q_i$, are then weighted by a coefficient $w'_r(p_i, p_c)$ and a coefficient $w'_t(q_i, q_c)$. The final aggregated cost is computed by summing

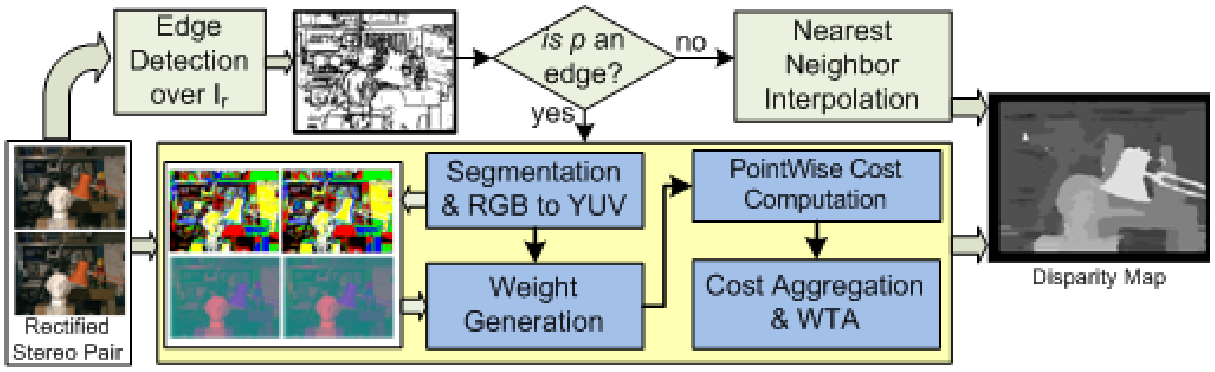


**Figure 1. Overall flow of the algorithm implemented by the proposed disparity estimation system.**

up all the weighted pointwise scores, and normalizing by the weights sum as in (2). The segmentation, weight generation, pointwise score computation and cost aggregation steps are executed for all disparity levels and the best disparity for the pixel *p* is found by locating the disparity with the minimum aggregated cost through a winner-takes-all (WTA) approach. In the case where pixel *p* does not correspond to an edge, the disparity is obtained by a simple nearest neighbor interpolation step.

$$w'_{r,t} = \begin{cases} 1.0 & p_i \epsilon S_c \\ exp\left(-\frac{d_c\left(I_{r,t}(p_i), I_{r,t}(p_c)\right)}{\gamma_c}\right) & otherwise \end{cases} \tag{1}$$

$$C(p_c, p_i) = \frac{\sum_{p_i \epsilon W_r, q_i \epsilon W_t} w'_r(p_c, p_i) \cdot w'_t(q_c, q_i) \cdot AD(p_i, q_i)}{\sum_{p_i \epsilon W_r, q_i \epsilon W_t} w'_r(p_c, p_i) \cdot w'_t(q_c, q_i)} \tag{2}$$

## 9.2 Hardware-Directed Optimization Techniques

The segment-based ADSW algorithm in [5] uses mean shift segmentation. However, the computational complexity and memory requirements make it unsuitable for embedded real-time applications. The k-means algorithm is simpler, and can be implemented in hardware more efficiently. In this work, we integrate an even simpler method, which partitions the image into segments using thresholding; we adopt this instead of the k-means, based on our observation that it has no negative impact on the overall disparity map accuracy. We also adopt YUV instead of CIELAB color representation in the weight generation step. This allows the use of unsigned integers instead of signed floating-point integers, which are complex and hardware-unfriendly. We also adopt Manhattan rather than Euclidean distance during the computation of the color distance between two YUV triplets. In this way, the square and square root operations are replaced by simple absolute difference and addition operations.

Furthermore, the *exp(-x)* function is approximated by the $\lfloor 2^{8-x} \rfloor$ function, which assigns a maximum weight of 256 if the color distance is zero and a weight of 0 if the color distance is greater than 8. This function simplifies the circuits that implement the multiplication of the weight coefficients with the pointwise scores, as multiplications are reduced to left shift operations. The cost function is further simplified by setting $\gamma_c$ to 16 instead of 22 (the value used in [5]). This converts the division to a right shift operation. Lastly, the denominator of (2) is approximated by the nearest power of 2 during the cost aggregation step, allowing the division to be replaced by a right shift operation.

We illustrate how the accuracy of the disparity maps is impacted by the aforementioned optimization techniques in Table I. We compare the error rate averaged over the Tsukuba, Venus, Cones and Teddy stereo images from Middlebury evaluation website [2] to a reference algorithm that works similarly with the one proposed in [5], but using the k-means instead of the mean shift segmentation. Moreover, the reference algorithm does not integrate any post-processing steps (for refining the disparity maps), as these are part of ongoing work. We observe that the overall error rate is reduced by ~2.9% after the integration of all optimization techniques.

## 9.3 Proposed Hardware Architecture

The proposed architecture consists of two major pipeline stages: the *Input Stage (IS)* and the *Calculation Stage (CS)*. The IS fetches pixel values from the input images in RGB format and performs pixel-based operations on them, such as RGB to grayscale conversion, RGB to YUV color conversion and segmentation. The image values computed by the IS are temporarily stored into on-chip buffers (memory arrangements - *MAs*), ensuring that there is always sufficient data for the CS, which is responsible for the calculation of the disparities. The overall system architecture also consists of a control unit that coordinates data transfers and handshakes between the different system units. Fig. 2 shows a block diagram of the proposed architecture and the data flow between units.

### 9.3.1 Input Stage (IS)

The IS consists of a memory controller, 2 RGB to YUV color space converters

TABLE I. IMPACT OF OPTIMIZATION TECHNIQUES ON ERROR RATE

| **Optimizations** | **1** | **1-2** | **1-3** | **1-4** | **1-5** | **1-6** | **1-7** |
|---|---|---|---|---|---|---|---|
| **Error rate** | 0% | -2.3% | -0.8% | -2.9% | -3.4% | -2.6% | -2.9% |

1 - Thresholding segmentation, 2 - YUV color representation, 3 – Manhattan color distance, 4 - $\lfloor 2^{8-x} \rfloor$ function, 5 – $\gamma_c = 16$, 6 – approximation of the denominator by the nearest power of 2, 7 – edge detector

(rgb2yuv), 2 RGB to grayscale converters (rgb2gray) and 2 segmentation units. The memory controller interfaces to an external memory and fetches the RGB color values corresponding to the support windows $W_r$ and $W_t$ in a column-wise fashion. Those values are then converted to grayscale by the two rgb2gray units, and to their corresponding 8-bit YUV representation by the rgb2yuv units. The image segmentation units receive the grayscale values and the number of segments $k$ given as input to the system (maximum supported $k$ is 32). A label (an unsigned integer in the range 1 to k) computed by a simple method that multiplies the input grayscale value by the value of k/256 is assigned to each input grayscale value. The multiplication is performed using fixed-point arithmetic with 8-bits of integer and 16-bits of fraction. The values of k/256 for all possible values of $k$ are stored in a look-up table (LUT). The result is given by taking the 5 most significant bits of the multiplication operation.

### 9.3.2 On-chip Memory Arrangements (MAs)

The on-chip MAs temporarily store the pixels required to perform correlation between $W_r$ in $I_r$ and the $d_M$ candidate support windows $W_t$ in $I_t$. The system is provided with 5 MAs per input image, which store the Y, U, V color values, the grayscale values and the segments. Fig. 3 (a-b) shows the architectures of the on-chip MAs for the reference and target image, respectively. Both MAs consist of a column buffer, a series of FIFO queues and window

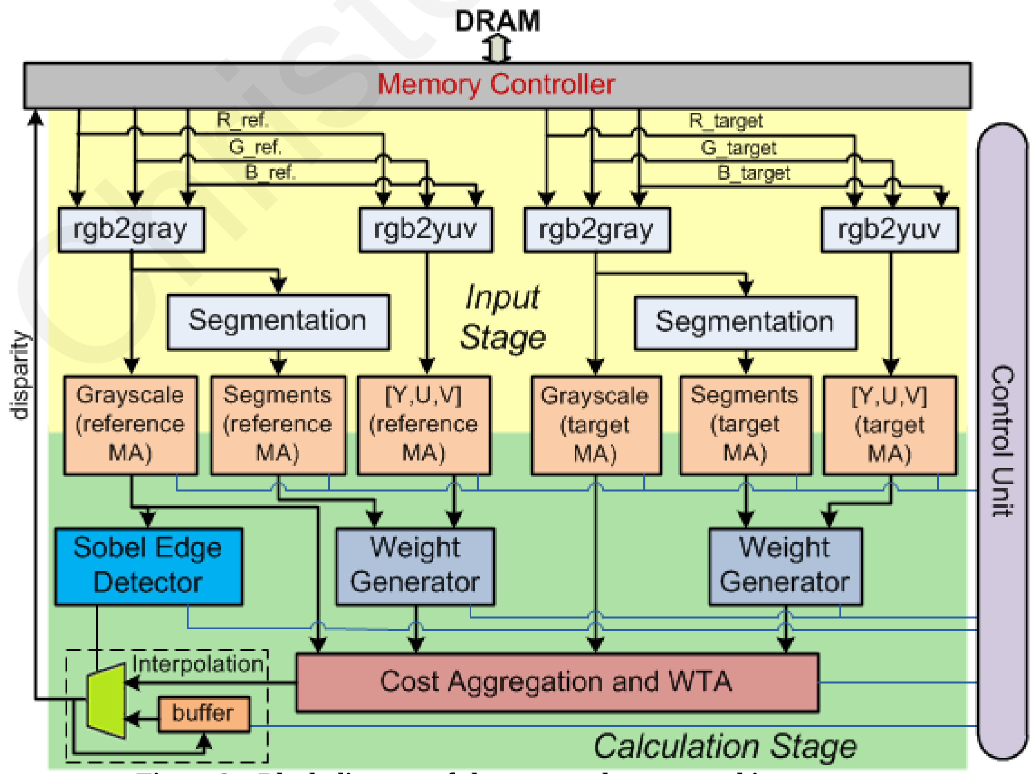


Figure 2. Block diagram of the proposed system architecture.

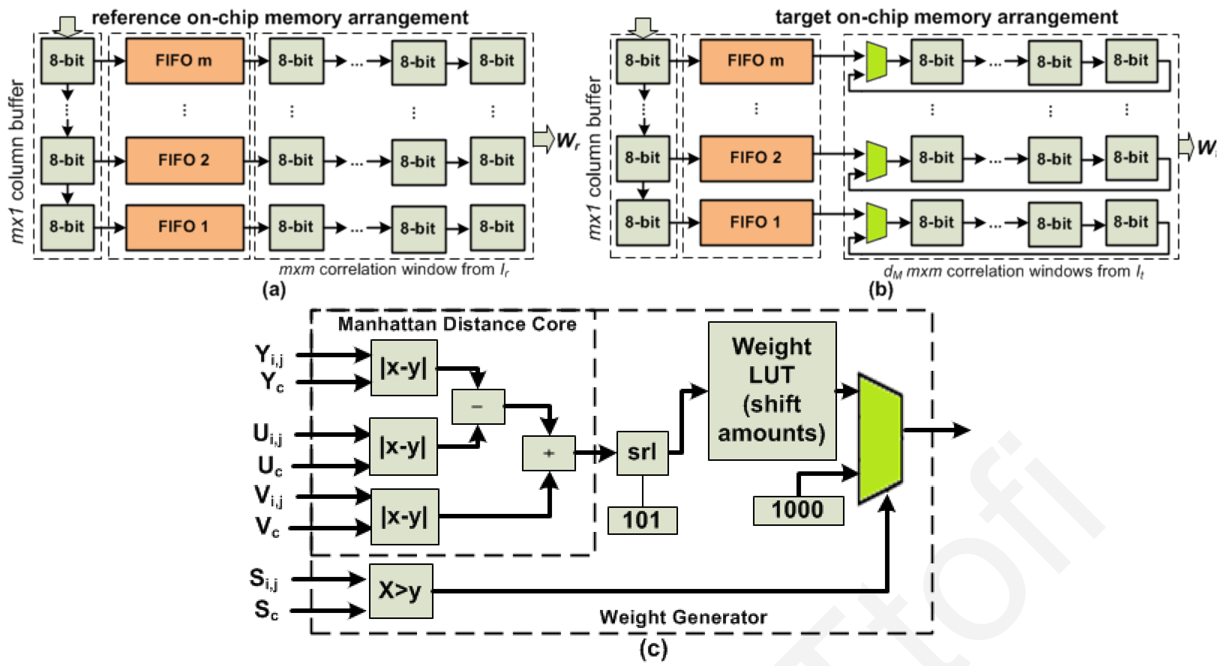


**Figure 3. (a)-(b) On-chip memory arrangements for the reference and target image data, (c) Circuit that computes the weight of a single pixel.**

buffers. The column buffer, which consists of *m* 8-bit registers, stores the pixels of an entire column of a support window. It receives one pixel per clock cycle and outputs a column every *m* cycles. The output column is stored in a series of *m* FIFO queues (1 pixel per queue), which are used to allow the memory controller to continuously fetch data from the external memory to the on-chip MAs (given that there is free space in the queues) irrespective of the data consumption rate of the CS. The CS consumes data at irregular rates; it consumes a column in $d_M$ cycles if *p* is an edge and in a single cycle if *p* is not an edge.

The data from the queues is forwarded to the window buffers, which form the inputs of the CS. The window buffer of the reference MA consists of *mxm* 8-bit registers, while the window buffer of the target MA consists of $m \cdot (m + d_M - 1)$ registers ($d_m$ is set to 0). The use of registers allows parallel access to the window buffers; after an initial delay of $m \cdot (m + d_M - 1)$ cycles per scanline (dominated by the cycles needed to fill in the window buffer of the target MA), both on-chip MAs can provide an *mxm* window per cycle. The window buffer of the target MA is organized in a cyclic structure, and is provided with a series of multiplexers at its input, which determine whether the input comes from the FIFO queues or from the rightmost registers of the window buffer. This structure is adopted to enable data reuse.

### 9.3.3 Calculation Stage (CS)

The CS consists of an edge detection unit, two units for the generation of the weight coefficients (weight generators), a unit that computes the aggregated costs and selects the disparity with the minimum cost, and a unit responsible for the nearest neighbor (NN) interpolation step.

#### 9.3.3.1 *Edge Detection Unit*

The edge detection unit implements a Sobel operator, which involves convolution of a 3x3 window (fetched from the reference MA that stores the grayscale values) using two convolution masks; the masks hold data values between -2 and 2; thus the overall convolution does not involve a multiplier. The result of the convolution is compared to a predetermined threshold. The comparison returns 1-bit pixel values that indicate whether the pixel being processed is an edge or not.

#### 9.3.3.2 *Weight Generator*

The weight generator computes the weight coefficients $w'_{r,t}$ for a support window $W_{r,t}$ in parallel. It receives the information about the segments and the YUV color values corresponding to the support window $W_{r,t}$ from the Segments and [Y,U,V] MAs, respectively, and computes the $m^2$ weight coefficients using $m^2$ instances of the circuit shown in Fig. 3 (c).

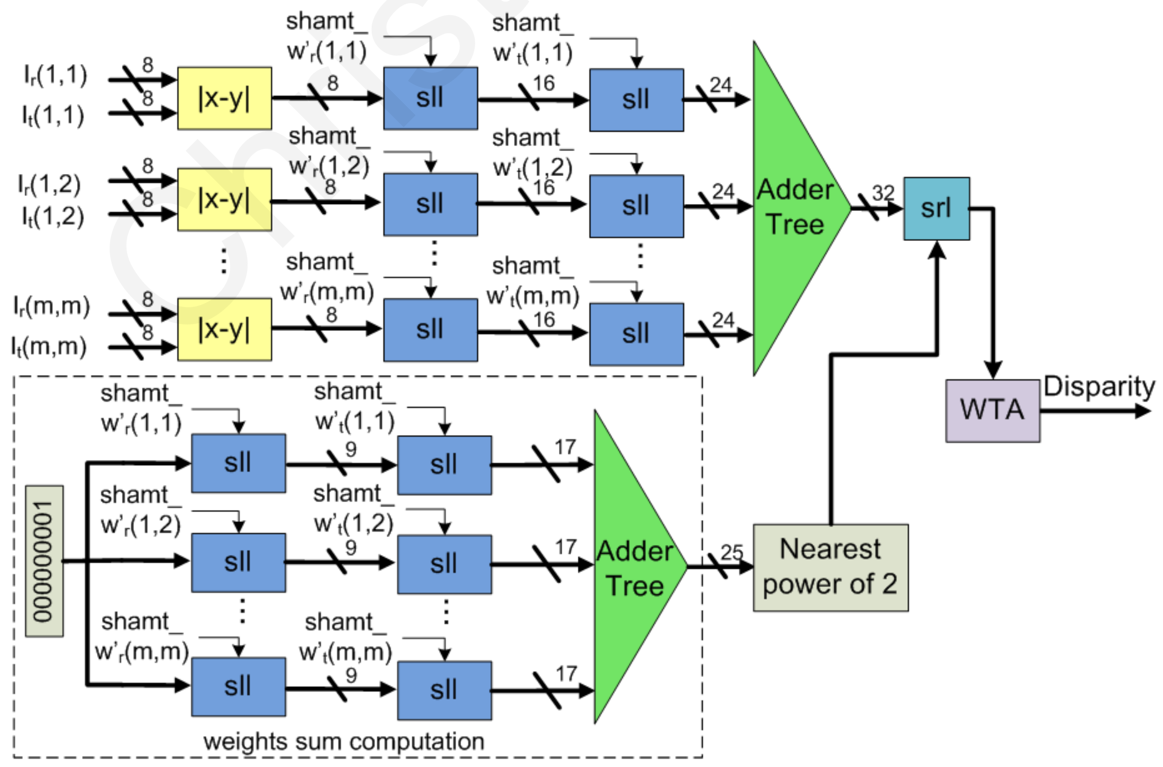


**Figure 4. Architecture of the cost aggregator and WTA.**

That circuit consists of a comparator, a Manhattan distance core and a weight table (LUT). Since the multiplication of the pointwise scores by the weight coefficients (cost aggregation step) is performed using shifters instead of multipliers, each location *x* of the LUT stores the shift amount corresponding to the weight coefficient $\lfloor 2^{8-x} \rfloor$. This shift amount is equal to the binary logarithm of $\lfloor 2^{8-x} \rfloor$, except from values of *x* greater than 8, for which a binary logarithm does not exist. In that special case, the corresponding entries in the LUT are set to a number, which is large enough so that the result of a shift operation by that number is equal to zero. The comparator determines whether the pixel at location *(i,j)* in $W_{r,t}$ lies in the same segment with the central pixel of $W_{r,t}$. The result of the comparator specifies whether the shift amount that corresponds to the weight coefficient $w'_{r,t}(i,j)$ will be assigned to the shift amount corresponding to the maximum weight (8 in our case) or whether it will be looked up in the LUT using the color distance generated by the Manhattan Distance core as index.

#### 9.3.3.3 *Cost Aggregation & Winner-Takes-All (WTA)*

The architecture of the cost aggregator and WTA unit is shown in Fig. 4. The unit utilizes $m^2$ absolute different circuits that compute the pointwise scores between corresponding pixels in $W_r$ and $W_t$. Those scores are then shifted by the shift amounts corresponding to the weight coefficients $w'_r$ and $w'_t$ using a series of left shifters (equivalent to multiplying the scores by $w'_r$ and $w'_t$). The final aggregated cost is computed by summing the outputs of the left shifters using a tree adder, and then normalizing (dividing) it by the weights sum, which, before being used for division, is rounded to the nearest power of 2 by using tree comparators. This enables a cost-effective implementation of the division using a right shifter. Finally, the WTA unit selects the disparity with the minimum cost.

#### 9.3.3.4 *Nearest Neighbor Interpolation*

CS integrates a simple and fast nearest neighbor interpolation unit, used to assign each pixel *p* at (x, y) not corresponding to an edge, to the disparity value of the nearest edge point (x', y) in the same scanline, where (x'< x).

## 9.4 Experimental Platform and Results

### 9.4.1 Experimental Platform

A prototype of the architecture shown in Fig. 2 was implemented on the Xilinx ML505 board, which features a Virtex-5 LX110T FPGA. The system was evaluated using rectified synthetic and real-world data, initially stored in the compact flash memory card. The synthetic

TABLE II. QUALITY COMPARISON OF DIFFERENT DEDICATED HARDWARE IMPLEMENTATIONS

| | Tsukuba | | | Venus | | | Teddy | | | Cones | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | nonocc | all | disc | nonocc | all | disc | nonocc | all | disc | nonocc | all | disc |
| **Georgoulas [14]** | n.a | 13.5 | n.a | n.a | 12.6 | n.a | n.a | n.a | n.a | n.a | 12.6 | n.a |
| **Darabiha [16]** | 19.5 | n.a | 37.6 | 10.51 | n.a | 31.5 | n.a | n.a | n.a | n.a | n.a | n.a |
| **Jin [17]** | 9.79 | 11.5 | 20.2 | 3.59 | 5.27 | 36.8 | 12.5 | 21.5 | 30.5 | 7.34 | 17.5 | 21 |
| **Ambrosch [18]** | 5.81 | 7.14 | 22.6 | 2.61 | 3.33 | 25.3 | 9.79 | 15.5 | 25.7 | 5.08 | 11.5 | 15 |
| **Chang [19]** | n.a | 2.80 | n.a | n.a | 0.64 | n.a | n.a | 13.7 | n.a | n.a | 10.1 | n.a |
| **Proposed** | 4.48 | 6.04 | 12.7 | 6.01 | 7.47 | 18.2 | 21.5 | 28.1 | 28.8 | 17.1 | 25.9 | 25.8 |

data includes stereo images from the Middlebury database [2], and the real-world data includes stereo images taken in the lab. The images were loaded into the on-board DRAM using the Microblaze soft-processor, and were used as input to the system shown in Fig. 2. The resulting disparity maps were directed to a TFT monitor. The evaluation results of the synthetic images are shown in Fig. 5 (a) and Fig. 5 (c) (column 3). The evaluation results of the real-world images are shown in Fig. 5 (c) (columns 1 & 2).

### 9.4.2 Disparity Map Quality Analysis

The quality of the generated disparity maps was evaluated quantitatively using Middlebury stereo pairs, and by measuring the incorrect disparity estimates using the percentage of bad pixels, a commonly accepted metric [2]. Results are given in Table II, which also compares the quality of our implementation to other existing hardware implementations.

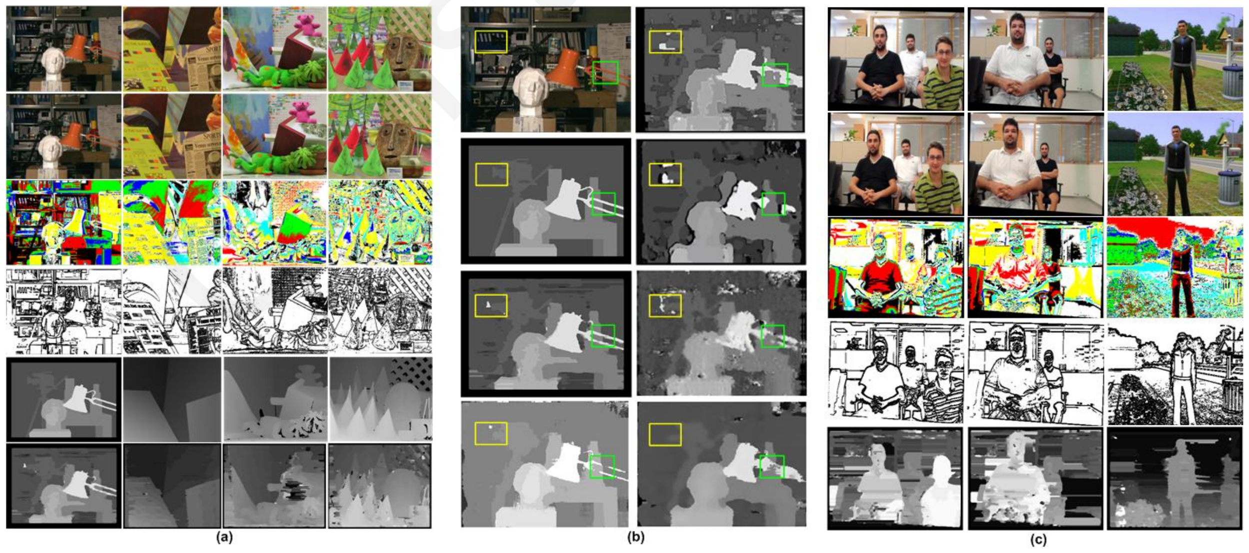


**Figure 5. (a) Evaluation results using Middlebury stereo pairs for a correlation window size of 13x13 (from left to right: Tsukuba, Venus, Teddy and Cones). From top to down: reference image, target image, output of the segmentation step for the reference image, output of the edge detector for the reference image, ground truth, disparity map of the proposed FPGA implementation (b) Disparity map of the Tsukuba image pair for different implementations. From top to down: reference stereo image, ground truth, proposed, Chang et. al., Georgoulas et. al., Jin et. al., Darabiha et. al., and Ambrosch et. al., (c) Evaluation results for real-world images (columns 1 & 2) and a computer-generated image (column 3).**

We omit results from [8]-[9], [11]-[13], [15] because they do not present quality results or because a different quality metric is used. For qualitative comparisons, we include the disparity maps generated by different implementations in Fig. 5 (b).

The proposed implementation yields better quality than the ones listed in Table II when considering the Tsukuba image pair, excluding the implementation in [19]. This can also be observed in Fig. 5 (b) and particularly at depth discontinuity regions (indicated with green), and at regions with repetitive patterns (indicated with yellow). Moreover, the percentage of bad pixels at depth discontinuity regions is lower than all other implementations when considering the Venus stereo image pair. In the Teddy and Cones images, the proposed implementation yields a comparable quality to [14] and [17], but a lower quality when compared to [18] and [19]. It must be noted that the proposed implementation, along with [19], are the only ones that implement an ADSW algorithm. However, [19] does not provide quantitative results for the error rate at depth discontinuity regions. These regions are significantly important when implementing ADSW algorithms, as the idea of an adaptive support window is primarily motivated by the need to accurately detect depth borders (where depth discontinuities occur). As such, the effectiveness of the implementation in [19] cannot be directly compared to the proposed implementation. In addition, [19] focuses only on synthetic image data; in contrast, we provide evaluation results for real-world data in Fig. 5 (c), evidencing the effectiveness of the proposed implementation. We anticipate that the proposed architecture can achieve even better quality by integrating post-processing steps such as left-to-right consistency check, sub-pixel estimation, spike removal and interpolation of the occluded regions [2], [17]; the majority of the implementations listed in Table II already implement these steps.

### 9.4.3 Processing Speed

We measured the processing speed (in fps) of the proposed hardware implementation using synthetic stereo image pairs from [2] as benchmarks. Tables III & IV illustrate how the frame rate is affected by the maximum disparity range and image size. We also give the frame rate of the system without the edge detector. The frame rate decreases as the disparity range and the image size increase. In both cases, the edge detector offers significant speedup (2.2 – 3.5) due to its ability to reduce the search data of the input images by an average of 55-85%, depending on the threshold used by the detector.

Table V presents a comparison between existing implementations and the proposed

TABLE III. PROCESSING TIME (FPS) VS. DISPARITY RANGE
*(Image Size = 320x240, Support Window Size = 13x13)*

| Disparity Range | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|
| fps w/ Edge Detector | 590 | 361 | 207 | 115 | 66 |
| fps w/o Edge Detector | 239 | 124 | 65 | 34 | 19 |

TABLE IV. PROCESSING TIME (FPS) VS. IMAGE SIZE
*(Max Disparity Range = 64, Support Window Size = 13x13)*

| Image Size | 100x100 | 240x160 | 320x240 | 640x480 | 800x600 |
|---|---|---|---|---|---|
| fps w/ Edge Detector | 811 | 227 | 115 | 30 | 19 |
| fps w/o Edge Detector | 336 | 71 | 34 | 8 | 5 |

architecture. Performance is provided in frames per second (fps), as well as in Million Disparity Estimations per second (MDE/s). The proposed system achieves 30 fps for an image size of 640x480 and a disparity range of 64. Such rates seem sufficient considering that the proposed architecture implements one of the most complex and accurate local stereo correspondence algorithms available in literature [20]. The majority of works listed in Table V implement simple and fast fixed support and multiple window algorithms; only [19] implements an ADSW algorithm. Even though the complexity of the algorithm implemented by the proposed architecture is twice as large as the complexity of the algorithm in [19] (as shown in [20]), the proposed system outperforms [19]. This is attributed to the reduction of the search space due to the integration of the edge detector.

### 9.4.4 Hardware Overheads

The proposed FPGA system was evaluated for relevant metrics such as area and frequency. Table VI gives the overall hardware demands of the FPGA prototype, which supports a window size of 13x13 and 64 disparity range levels. The system utilizes ~90% of the FPGA LUTs and ~80% of the FPGA slice registers, and can operate at 155MHz. The slice

TABLE V. PROCESSING SPEED COMPARISON FOR VARIOUS SYSTEMS

| Work | Image size | Disparity Range | Frame rate (fps) | PDS ($10^6$ ) | Freq. (MHz) |
|---|---|---|---|---|---|
| Hile [8] | 512x480 | 32 | 30 | 235.9 | n.a. |
| Miyajima [9] | 640x480 | 80 | 26 | 639 | 40 |
| *Arias-Estrada* [11] | 320x240 | 16 | 71 | 87.2 | 66 |
| Lee [12] | 640x480 | 64 | 30 | 589 | 10 |
| Hariyama [13] | 64x64 | 64 | 5063 | 1327.2 | 86 |
| Georgoulas [14] | 800x600 | 80 | 550 | 21120 | 511 |
| Ambrosch [15] | 450x375 | 100 | 600 | 10125 | 110 |
| Darabiha [16] | 256x360 | 20 | 30.3 | 55.2 | n.a. |
| Jin [17] | 640x480 | 64 | 230 | 4522 | 93.1 |
| Ambrosch [18] | 750x400 | 60 | 60 | 1080 | 133 |
| Chang [19] | 352x288 | 64 | 42 | 272.5 | n.a. |
| Proposed | 640x480 | 64 | 30 | 589 | 155 |

TABLE VI. COMPLETE SYSTEM HARDWARE OVERHEADS
*(Image Size=640x480, Max Disparity Range=64, Support Window Size=13x13)*

| Platform | Slice LUTs (69120) | Slice Registers (69120) | DSP48Es (64) | BRAMs (140) | Freq. (MHz) |
|---|---|---|---|---|---|
| Virtex-5 LX110T FPGA | 62213 (~90.01%) | 55594 (~80.43%) | 16 (~25.0%) | 30 (~20.3%) | 155 |

LUTs are dominated by the cost aggregator and the weight generators, which consume ~64% of the available LUTs. The slice registers are dominated by the on-chip MAs, which consume ~35% of the available slice registers.

## 9.5 Conclusion

This appendix section discussed how the edge-directed search space reduction approach (Section 3) can be integrated within a segmentation-driven ADSW stereo matching algorithm. Through the utilization of edge information and the integration of a series of hardware-oriented design optimizations, the complex ADSW-based matching algorithm has been adopted for an efficient real-time design, which provides high disparity map quality when compared to other implementations featuring fixed support stereo correspondence algorithms.

## 9.6 References

[1] B. Cyganek, J. P. Siebert, *Introduction to 3D Computer Vision Techniques and Algorithms*, Wiley, John & Sons, 2009.

[2] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *Inter. J. of Comput. Vision*, vol. 47, pp. 7–42, 2002.

[3] M. Z. Brown, D. Burschka, G. D. Hager, and S. Member, “Advances in computational stereo,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, pp. 993–1008, 2003.

[4] K.-J. Yoon and I.-S. Kweon, “Adaptive support-weight approach for correspondence search,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650–656, Apr. 2006.

[5] F. Tombari, S. Mattoccia, and L. Di Stefano, “Segmentation-based adaptive support for accurate stereo correspondence,” *in Lecture Notes in Computer Science*, vol. 4872, Berlin, Germany: Springer, Dec. 2007, pp. 427–438.

[6] S. Hadjitheophanous, C. Ttofis, A. S. Georghiades, T. Theocharides, “Towards hardware stereoscopic 3D reconstruction: A real-time FPGA computation of the disparity map,” *Design, Automation & Test in Europe Conference & Exhibition 2010 (DATE’10)*, pp.1743-1748, Dresden, Germany, 8-12 March 2010.

[7] M. Gerrits and P. Bekaert, “Local stereo matching with segmentation-based outlier rejection,” *in Proc. 3rd Canadian Conf. Comput. Robot Vision*, Jun. 2006, p. 66.

[8] H. Hile, C. Zheng, Stereo Video Processing for Depth Map, Technical Report, University of Washington, 2004.

[9] Y. Miyajima and T. Maruyama, “A Real-Time Stereo Vision System with FPGA,” *13th Int. Conf. on Field-Programmable Logic and Applications*, Lisbon, Portugal, 2003, pp. 448-457.

[10] L. Nalpantidis, G. Sirakoulis, A. Gasteratos, "Review of Stereo Matching Algorithms for 3D Vision,"*Proc. of the 16th Int. Symposium on Measurement and Control in Robotics (ISMCR 2007)*, Poland, 21-23 June 2007, pp. 116-124.

[11] M. Arias-Estrada, J. M. Xicotencatl, "Multiple Stereo Matching Using an Extended Architecture", *Field-Programable Logic and Applications*, vol. 2778, Springer Berlin/Heidelberg, 2003, pp. 203-212.

[12] S. H. Lee, J. Yi, J. Kim, "Real-Time Stereo Vision on a Reconfigurable System," *Embedded Computer Systems: Architectures, Modeling and Simulation*, vol. 3553, Springer Berlin/Heidelberg, 2005, pp. 299–307.

[13] M. Hariyama, Y. Kobayashi, H. Sasaki, M. Kameyama, "FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture, *48th Midwest Symp. on Circuits and Syst.*, vol. 2, Covington, KY, 2005, pp. 1219 –1222.

[14] C. Georgoulas, I. Andreadis, "A Real-Time Occlusion Aware Hardware Structure for Disparity Map Computation," *Image Analysis and Process. – ICIAP 2009*, vol. 5716/2009, pp. 721-730.

[15] K. Ambrosch, M. Humenberger, W. Kubinger, A. Steininger, "A SAD-based Stereo Matching Using FPGAs," *Embedded Computer Vision part II*, pp. 121-138, Spinger, London (2009).

[16] A. Darabiha, J. MacLean, J. Rose, "Reconfigurable hardware implementation of a phase-correlation stereo algorithm," *Machine Vision Appl.*, vol. 17, no. 2, pp. 116–132, 2006.

[17] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S-K. Park, M. Kim, J. W. Jeon, "FPGA Design and Implementation of a Real-Time Stereo Vision System," *IEEE Trans. Circuits Syst. Video Technol.*, pp. 15 - 26, Jan. 2010.

[18] K. Ambrosch, W. Kubinger, "Accurate hardware-based stereo vision," *Computer Vision and Image Understanding*, pp. 1303-1316, 2010.

[19] N.Y.-C. Chang, T-H Tsai, B-H Hsu, Y-C Chen, T-S Chang, "Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight," *IEEE Trans. Circuits Syst. Video Technol.*, vol.20, no.6, pp.792-805, June 2010.

[20] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Classification and evaluation of cost aggregation methods for stereo correspondence," *IEEE Int. Conf. Comput. Vision Pattern Recognit.*, Jun.2008, pp. 1–8.

# Biography

Christos Ttofi received the Bachelor's Degree and Master's Degree in Computer Engineering from the University of Cyprus in 2009 and 2011, respectively. Since 2011 he is working toward the PhD degree in Computer Engineering at the University of Cyprus. He is a researcher at the Embedded and Application Specific System-on-Chip Laboratory (EASoC) at the KIOS Research Center. His research interests include Embedded Systems Design, Image Processing and Computer Vision, Field Programmable Gate Arrays (FPGAs), Computer Arithmetic and Computer Architecture. Christos has been involved in various projects funded by the European Commission and the Research Promotion Foundation of Cyprus. He is also a student Member of the IEEE and the IEEE Computer Society, and a member of the Technical Chamber of Cyprus (ETEK).